Developing Microservices with webMethods Microservices Runtime

Version 10.11

October 2021

WEBMETHODS

This document applies to webMethods Integration Server 10.11 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2007-2024 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at https://softwareag.com/licenses/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at https://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

Document ID: WMIC-DG-1011-20240709

Table of Contents

About this Guide

This guide provides information about administering webMethods Microservices Runtime as well as functionality that is specific to Microservices Runtime.

Note:webMethods Microservices Runtime provides a superset of the functionality available in webMethods Integration Server. An Integration Server can be equipped with a Microservices Runtime license to use the features and functionality specific to Microservices Runtime. For information about administering and using features in webMethods Integration Server, see the *webMethods Integration Server Administrator's Guide*.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at https://documentation.softwareag.com.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at https://techcommunity.softwareag.com. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://hub.docker.com/u/softwareag and discover additional Software AG resources.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Getting Started with webMethods Microservices Runtime

What Are Microservices?

Microservices are independently deployable units of logic in which each microservice performs a single business function. Applications built in the microservices architectural style are developed as a suite of microservices.

Microservices can be implemented in various ways, including as a set of services or as event and channel definitions. Microservices can be distinguished from other types of services in the webMethods suite as follows:

The following table identifies and describes the types of services in the webMethods suite.

Type of Service	Description
Integration Server service	Single function call, such as an operation on an interface or as part of an API (that is, a related set of Integration Server services).
Web service	Formal, WSDL-based grouping of operations, usable for SOA. Many of these can be hosted in a single runtime.
Microservice	Collection of operations implemented as services or as messages and message handlers, deployed in one or more related Integration Server packages.

What Is webMethods Microservices Runtime?

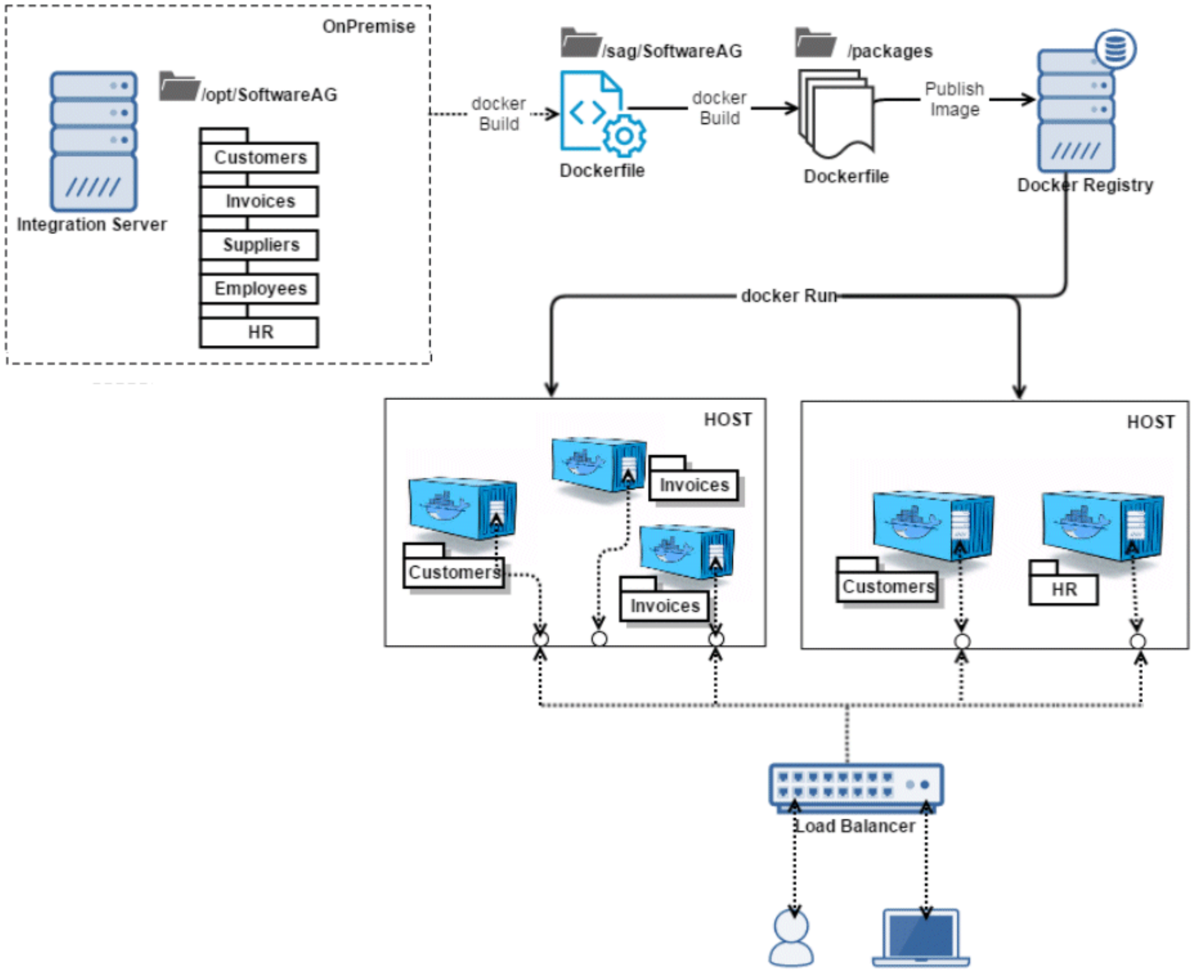
Software AG offers a lightweight container called webMethods Microservices Runtime to host microservices you develop in Software AG Designer. Using Microservices Runtime, you can deliver microservices as an Integration Server package that includes a set of related services, interfaces, document types, and triggers that subscribe to topics or queues, or as a set of related packages of this kind (for example, five packages relating to Human Resources functions).

Each microservice can run in its own Microservices Runtime and can communicate with lightweight mechanisms such as an HTTP resource API. However, you can also execute multiple microservices in the same Microservices Runtime. This hybrid solution enables you to separate microservices when needed, but also to group them when necessary. For example, suppose that you have two microservices that need to be scaled together in similar ways (that is, when you need a new instance of one, you need a new instance of the other). If you discover that one microservice is more heavily loaded than the other, or needs to be enhanced or updated more often, you could deploy the two microservices to separate Microservices Runtimes. If both microservices tend to be updated at the same time, you could cohost them in the same Microservices Runtime.

Microservices Runtime is fully compatible with webMethods Integration Server and can also host services you develop using Software AG Designer and Integration Server. While Microservices Runtime is optimized to have a reduced disk and memory footprint, you can convert it into a full Integration Server by installing additional modules, such as support for an external database.

Microservices Runtime provides out-of-the-box support for dynamic lookup of service endpoints using the open-source service registry named Consul. You can make your microservice available for remote access by registering the endpoint of the microservice container instance in the Consul service registry. Microservices Runtime provides services for automating the registration and deregistration process. Microservices Runtime provides facilities to look up the endpoint information which can be used to call the microservice at run time. You can also create your own package integrating with any other service registry provider. The package would provide services similar to those described for Consul in this guide.

Microservices Runtime is optimized for execution in a Docker container. You can run a microservice or a set of related microservices in a Docker container, precluding the need to purchase expensive virtual machines. Docker images include configuration, enabling you to deploy the exact same configuration anywhere. The Docker image can include one package or a set of related packages.



What Is Microservices Runtime Administrator?

Microservices Runtime Administrator is an HTML-based utility you use to administer Microservices Runtime. It allows you to monitor server activity, manage user accounts, make performance adjustments, and set operating parameters.

You can run the Microservices Runtime Administrator from any browser-equipped workstation on your network. Microservices Runtime Administrator is a browser-based application that uses services to accomplish its work.

2 Starting, Shutting Down, and Restarting Microservices Runtime

Starting Microservices Runtime and Microservices Runtime Administrator

Microservices Runtime must be running in order for clients to execute services or for Microservices Runtime to send outbound requests. If you are using Microservices Runtime in a development environment, it must be running in order for your developers to build, update, and test services using the Software AG Designer.

Before starting Microservices Runtime, make sure there is enough free disk space on the host machine to accommodate the storage of configuration and log files on disk. Running out of disk space can affect performance and lead to errors.

Note:
During installation of Microservices Runtime, an option can be set requiring that the password for the Administrator user account be changed immediately upon successful log in to Microservices Runtime. If so, the Administrator user account password must be changed before the server can do anything else. Any requests sent to Microservices Runtime before the Administrator password is changed will be rejected.

For information about starting Microservices Runtime in safe mode, see the *webMethods Integration Server Administrator's Guide*.

Starting Microservices Runtime on Windows

To start Microservices Runtime on Windows

- Click **Start > All Programs > Software AG > Start Microservices Runtime**.

Starting Microservices Runtime on UNIX

To start Microservices Runtime on UNIX

1. Navigate to the *Integration Server_directory* /bin directory and run the startup.sh or server.sh script file.

2. If Microservices Runtime has been configured to request a master password for outbound password encryption, you will be prompted for this password in a popup window or from the server console. For information about managing outbound passwords, see the *webMethods Integration Server Administrator's Guide*.

Starting Microservices Runtime from the Command Line

You can start Microservices Runtime from the command prompt. Starting Microservices Runtime this way gives you the option of overriding certain settings in the configuration file.

To start Microservices Runtime from the command line

1. At the command line, type the following to switch to Microservices Runtime's home directory:

   cd *Integration Server_directory*

2. Type one of the following commands to start the server.

   For Windows: `bin/startup.bat - switch - switch ...`

   For UNIX: `bin/startup.sh -switch -switch ...`

   where `-switch` is optional and can be one of the following:

Switch	Description
`-port portNumber`	Specifies the port on which the server listens for HTTP requests. *portNumber* specifies the TCP/IP port number Example: `-port 8080` This switch overrides the value assigned to watt.server.port. **Note:** In addition to overriding the value of watt.server.port, the `-port` switch permanently adds a new HTTP port to the WmRoot package. This new port is added as the primary port and contains default values. If a port with the same TCP/IP number already exists in the WmRoot package, the `-port` switch overrides its settings with the new default values. In effect, deleting the existing port and then adding a new port with default settings. **Note:** To use port 80 (the standard for HTTP) or port 443 (the standard for HTTPS), UNIX users must be running as "root". For security reasons, a better method is to use a higher number port (5555 for HTTP and 5543 for HTTPS), and if necessary have the firewall remap port 80 to the desired port.
`-secureport portNumber`	Specifies the port number for the DefaultSecure (HTTPS) port. *portNumber* specifies the TCP/IP port number Example: `-secureport 4355` This switch overrides the value assigned to watt.server.securePort. **Note:** In addition to overriding the value of watt.server.securePort, the `-secureport` switch permanently adds a new HTTPS port to the

Switch	Description
	WmRoot package. This new port is added as the DefaultSecure port and contains default values.
`-debug` *`level`*	Specifies the level of detail you want the server to maintain in its server log for this session. *level* indicates the level of detail you want to record in the log. **Specify...** / **To record...** `Fatal` — Fatal messages only. `Error` — Error and fatal messages. `Warn` — Warning, error, and fatal messages. `Info` — Informational, warning, error, and fatal messages. `Debug` — Debug, informational, warning, error, and fatal messages. `Trace` — Trace, debug, informational, warning, error, and fatal messages. For this session, this switch overrides the value specified for the Default facility on the **Logs > Logging configuration > View server logger details** page and assigned to watt.debug.level.
`-log` *`destination`*	Specifies where you want the server to write its server log information for this session. Specify one of the following for *destination*: **Option** / **Description** *filename* — Specify the fully qualified path or relative path to the file in which you want the server to write server log information for this session. Relative path is relative to the Microservices Runtime home directory. The *filename* must specify a directory and filename. The default destination is controlled by the watt.debug.logfile server configuration parameter. `none` — Write server log information to the computer screen (STDOUT).

Switch	Description
	`both` — Write server log information to the computer screen (STDOUT) and to the destination specified by the watt.debug.logfile parameter. When setting `-log both`, server log messages written to STDOUT include the identifier “ISSERVER” to help differentiate server log messages from other messages written to the console. The *filename* and `none` values override the value assigned to watt.debug.logfile for this session. For UNIX, setting `-log` to `none` or `both` results in the redirection of STDOUT to the server.out file located under *Integration Server_directory* /bin/logs **Note:** A `-log` switch value of `none` or `both` also determines where Microservices Runtime writes the configuration variables log. However, Microservices Runtime ignores a filename value. If you specify a filename, Microservices Runtime writes the configuration variables log file to this location only: *Integration Server_directory* /i/logs/configurationvariables.log. For more information about configuration variable templates and the associated logging, see “Using Configuration Variables Templates with Microservices Runtime ” on page 45
`-logformat` *`type`*	Specifies the format to which Integration Server converts the server log and audit log entries written to the console (STDOUT or computer screen). Specify 'type' as json. Integration Server supports only "json" type. To send the audit log entries in the JSON format to the console, in addition to setting the -logformat switch, set the SAG_IS_AUDIT_STDOUT_LOGGERS environment variable to a comma-separated list of the audit loggers that log entries to STDOUT. For example, WMSESSION, WMERROR. For more information, see *webMethods Integration Server Administrator’s Guide*. **Note:** This switch applies when you start Integration Server with the `-log` *`both`* switch. Using the `-logformat` switch does not affect log files such as server.log and error.log.
`-quiesce`	Specifies to start the server in quiesce mode.

Switch	Description
	Note: If the server cannot disable or suspend an asset or activity as it enters quiesce mode, the server displays a warning message and continues the quiesce process. If the condition stated in the warning interferes with a maintenance task you intend to perform, resolve the issue stated in the warning and then restart the server in quiesce mode.

Running Microservices Runtime in Debug Mode

If you want Microservices Runtime to run in debug mode, you can run the startDebugMode.bat(sh) script located in *Integration Server_directory* /bin/. Running in debug mode allows you to connect an Eclipse-based Debugger to your running Microservices Runtime.

The default port used for debug mode is 10033. Additionally, by default, the JVM waits for Debugger to attach before the actual startup of Microservices Runtime. The DEBUG_PORT and SUSPEND_MODE properties in the startDebugMode.bat(sh) script control these defaults. You can change the debug port and suspend mode for a session by starting debug mode from the command line using a command like the following:

```
startDebugMode.bat address 8787 suspend n
```

Where `address` refers to the port used for DEBUG_PORT (default is 10033) and `suspend` indicates whether to wait for Debugger to attach to the JVM (default is y).

Starting Microservices Runtime Administrator

To start Microservices Runtime Administrator

1. Do one of the following:

   - Open a browser and point it to the host and port where Microservices Runtime is running (for example, `http://localhost:5555` or `http://EXAMPLE:4040`).
   - On Windows, click **Start > All Programs > Software AG > Administration > Microservices Runtime Administrator**.

2. Log on to Microservices Runtime Administrator with a user name and password that has administrator privileges. User names and passwords are case sensitive.

   If this is the first time the Administrator user is logging in to Microservices Runtime Administrator, keep the following points in mind:

   - During installation of Microservices Runtime, the person performing the installation specifies a password for the Administrator user. You must supply this password on initial log in to Microservices Runtime Administrator.

- The person performing the installation might also set an option requiring that the Administrator password be changed immediately upon successful log in to Microservices Runtime Administrator. If so, Microservices Runtime Administrator displays the Create new password dialog box after the Administrator logs in for the first time.

Important:
Use the *exact combination* of upper and lowercase characters as user names and passwords are case sensitive.

Shutting Down Microservices Runtime

When you shut down Microservices Runtime, all active sessions also shut down. for instructions on viewing active sessions before shutting down, see *webMethods Integration Server Administrator's Guide*.

Shutting Down Microservices Runtime on Windows

To shut down Microservices Runtime on Windows

- Click **Start > All Programs > Software AG > Stop Microservices Runtime**

Shutting Down Microservices Runtime from Microservices Runtime Administrator on Windows or UNIX

To shut down Microservices Runtime from Microservices Runtime Administrator on Windows or UNIX

1. In Microservices Runtime Administrator, click ⏻ **> Shut down or restart**.

2. Specify whether you want you want Microservices Runtime to wait before shutting down or shut down immediately.

To	Do this
Shut down after *number* minutes or after all client sessions are complete	Select **After all client sessions end**. Then in the **Maximum wait time** field, specify the number of minutes you want Microservices Runtime to wait before restarting. Microservices Runtime will begin monitoring user activity and then automatically shut down after all non-administrator sessions complete or after the time you specify elapses, whichever comes first.
Shut down immediately	Select **Immediately**. Microservices Runtime and all active sessions will terminate immediately.

3. Click **Shut Down**.

Shutting Down Microservices Runtime from the Command Line

Use this procedure to shut down Microservices Runtime and all active sessions from the command prompt.

To shut down Microservices Runtime from the command prompt

1. At the command line, type the following to switch to Microservices Runtime's home directory:

   cd *Integration Server_directory*

2. Type the following command to stop Microservices Runtime.

   For Windows: `bin\shutdown.bat`

   For UNIX: `bin/shutdown.sh`

Restarting Microservices Runtime

You should restart Microservices Runtime when:

- **You make certain configuration changes.** Some configuration changes require Microservices Runtime to be restarted before they take effect. Microservices Runtime Administrator displays a notification when a configuration change was made that requires a server restart for the change to take effect.
- **You want to incorporate updated services that cannot be dynamically reloaded.** This typically occurs for non-Java services.

To restart Microservices Runtime

1. In Microservices Runtime Administrator, click ⏻ **> Shut down or restart**.

2. Specify whether you want Microservices Runtime to wait before restarting or restart immediately.

To	Do this
Restart after *number* minutes or after all client sessions are complete	Select **After all client sessions end**. Then in the **Maximum wait time** field, specify the number of minutes you want Microservices Runtime to wait before restarting. Microservices Runtime will begin monitoring user activity and then automatically restart after all non-administrator sessions complete or after the time you specify elapses, whichever comes first.

To	Do this
Restart immediately	Select **Immediately**. Microservices Runtime and all active sessions will terminate immediately. Then Microservices Runtime restarts.

3. Click **Restart**.

3 Configuring Microservices Runtime

In general, configuring Microservices Runtime is the same as configuring Integration Server. However, there are differences between Integration Server and Microservices Runtime that result in differences in how each is configured. This topic provides more information about those differences.

For information about configuring Integration Server, see the *webMethods Integration Server Administrator's Guide*.

How Is Microservices Runtime Different from Integration Server?

Microservices Runtime is a superset of Integration Server which means that Microservices Runtime provides features and functionality that Integration Server does not. However, in addition to these features, Microservices Runtime differs from Integration Server in the following ways:

- Microservices Runtime does not support running multiple instances of Microservices Runtime on the same host machine under the same *Software AG_directory* . As such, the Microservices Runtime directory structure does not include the following directories:

  Integration Server_directory /instances/*instanceName*

  When reviewing documentation, references to *Integration Server_directory* /instances/*instanceName* for Integration Server map to the *Integration Server_directory* directory in Microservices Runtime.

- Microservices Runtime does not use the OSGi platform. Consequently, a Microservices Runtime installation does not include the following directory: *Integration Server_directory* /profiles/

- Microservices Runtime does not use the Java Service Wrapper developed by Tanuki Software, Ltd., Consequently, Microservices Runtime does not use the Java Service Wrapper configuration files wrapper.conf and custom_wrapper.conf for property settings. Instead, Java system properties are set in *Integration Server_directory* /bin/setenv.bat(sh) using the JAVA_CUSTOM_OPTS property.

- A JMX port is not enabled by default for Microservices Runtime.

- Microservices Runtime cannot be run as a Windows service.

- Features available on a standard Integration Server are optional on a Microservices Runtime. Optional feature are not installed by default on Microservices Runtime and must be selected specifically during the installation of Microservices Runtime.

Note:
The information above applies to a Microservices Runtime only. That is, the information does not apply to an Integration Server equipped with a Microservices license. For a detailed comparison of the functionality available in Integration Server vs. Microservices Runtime, see " Microservices Runtime vs Integration Server " on page 81.

Specifying the JDK or JRE for Microservices Runtime

Microservices Runtime must point to a Java location. By default, Microservices Runtime points to the location of the JRE installed at the same time you installed Microservices Runtime. If necessary, you can specify a different location.

Before you specify the location of Java for Microservices Runtime, determine whether you need to specify the location of the JDK or the JRE. If you intend to use Microservices Runtime to develop and compile Java services on Microservices Runtime, specify the location of the JDK. If you will not be using this installation of Microservices Runtime to compile Java services, you can specify the location of a JRE.

To specify the Java location for Microservices Runtime

1. Navigate to *Integration Server_directory* /bin and use a text editor to open the setenv.bat/sh file.

2. Set the JAVA_DIR parameter so that it specifies the location of the JDK or JRE installation directory.

3. Save and close the file.

4. Restart Microservices Runtime.

Note:
If you change the Java location and you use Microservices Runtime to develop and compile Java services, you must also change the value of the watt.server.compile configuration parameter.

Changing the JVM Heap Size Used by Microservices Runtime

The JVM heap or *on-heap* size indicates how much memory is allotted for server processes. At some point, you might need to increase the minimum and maximum heap size to ensure that the JVM that Microservices Runtime uses does not run out of memory. You will want to consider the heap size when you configure your server to publish and subscribe to documents and when you configure an on-heap cache.

The heap size is controlled by the following Java parameters specified in the setenv.bat/sh file.

Parameter	Description
JAVA_MIN_MEM	The minimum heap size. The default value is 256 MB.
JAVA_MAX_MEM	The maximum heap size. The default value is 1024 MB.

Your capacity planning and performance analysis should indicate whether you need to set higher maximum and minimum heap size values.

Note:

You can change the heap size for Microservices Runtime at start up by starting Microservices Runtime from the command line and supplying the `JAVA_MAX_MEM` and `JAVA_MIN_MEM` environment variables. Using the environment variables allows you to override the values set for Microservices Runtime in the *Integration Server_directory* /bin/setenv.bat|sh file without having to edit the file itself.

To change the heap size for Microservices Runtime

1. Navigate to *Integration Server_directory* /bin and use a text editor to open the setenv.bat/sh file.

2. Set the JAVA_MIN_MEM and JAVA_MAX_MEM parameters so that they specify the minimum and maximum heap size required by Microservices Runtime.

3. Save the file.

4. Restart Microservices Runtime.

Passing Java System Properties to Microservices Runtime

You can pass Java system properties to Microservices Runtime by modifying the setenv.bat/sh file.

To pass Java system properties to Microservices Runtime

1. Navigate to *Integration Server_directory* /bin and use a text editor to open the setenv.bat/sh file.

2. Set the JAVA_CUSTOM_OPTS parameter so that it specifies the property name and value you want to pass to Microservices Runtime. The property name must be preceded by `-D`.

   For example, the JAVA_CUSTOM_OPTS parameter in the setenv.bat file could look similar to the following:

   ```
   set JAVA_CUSTOM_OPTS="-Dmy.prop1=value1 -Dmy.prop2=value2"
   ```

3. Save the file.

4. Restart Microservices Runtime for the changes to take effect.

Enabling Remote Client JMX Monitoring

Microservices Runtime enables you to use JMX monitoring from a remote client. You enable JMX monitoring and set the JMX monitoring remote port in the setenv.bat/sh file of the Microservices Runtime you want to monitor. Unlike Integration Server, Microservices Runtime does not enable JMX monitoring by default.

To enable remote client JMX monitoring on Microservices Runtime

1. On the Microservices Runtime that you want to monitor remotely, navigate to *Integration Server_directory* /bin and use a text editor to open the setenv.bat/sh file.

2. Set the JMX_ENABLED property to `true`.

3. Set the JMX_PORT property to the port number of the JMX port. This is `9192` by default.

4. Restart Microservices Runtime.

Configuration of Additional Components

The following table identifies additional components that you might have installed with Microservices Runtime and provides the name of the guide that contains more information about configuring the component.

Component	**See this guide for more information**
CentraSite Asset Publisher Support	*webMethods Service Development Help*
Common Directory Service Support	*webMethods Integration Server Administrator's Guide*
External RDBMS Support	*webMethods Integration Server Administrator's Guide*

4 Using a Circuit Breaker with Services

About Circuit Breaker

Circuit breaker is an established design pattern that applications implement to prevent a failure in one part of the system from cascading to the rest of the system. In an architecture with distributed applications, such as microservices, many services call other services running on remote servers. If the remote service is unavailable or the network is slow, the calling service may wait until a timeout occurs. During this time, the calling service continues to consume critical resources such as server threads and memory. If multiple services call the unresponsive or failing remote service, the impact of the remote service cascades throughout all the calling services, causing even more resources to be consumed and affected by a single failing service. Implementing a circuit breaker on the call to the remote service can prevent the impact of the failing or unresponsive service or network latency from cascading throughout the system.

The circuit breaker design pattern works much like an electrical circuit breaker which is intended to “trip” or open the circuit when failure is detected. This prevents the flow of electrical current through the circuit. After a time delay, the electrical circuit breaker resets and closes the circuit, which causes the flow of electricity to resume.

In a software application, a circuit breaker functions as a proxy that executes the remote services and monitors the remote service for failures. A failure can be an exception and/or a timeout. When the number of failures meets a predetermined threshold within a specified time period, the circuit breaker “trips” or opens the circuit. Subsequent requests for the service end with an error or result in execution of an alternative service. After a reset period elapses, the circuit breaker sets the circuit state to half-open and executes the next request for the service. By allowing a single request to execute and causing other requests to wait, the circuit breaker gauges the health of the service. Upon success of the service, the circuit breaker closes circuit and waiting requests proceed. However, if the service ends with another failure, the circuit breaker re-opens the circuit.

Circuit breakers can be especially useful in systems with a microservices architecture as these systems often feature a large number of distributed components. By configuring a circuit breaker on the invocation of the remote service, you can limit the impact the abnormal behavior of a remote service on other micorservices and critical resources in your system.

Note:
The circuit breaker feature is available by default for a service that resides in a Microservices Runtime. To use the circuit breaker feature with Integration Server, your Integration Server must have additional licensing.

Circuit States

The state of a circuit for a service determines how circuit breaker responds to a request to invoke the service.

The following table identifies how the circuit breaker responds to an invocation request based on the circuit state.

If the circuit state is	Then
Closed	The circuit breaker executes the service.

If the circuit state is	Then
Open	The circuit breaker does not execute the service. Instead, the circuit breaker returns an exception, which allows the request to fail immediately, or circuit breaker invokes an alternative service.
Half-open	The circuit breaker executes the service the next time it is requested. If the service executes successfully, then the circuit breaker changes the circuit state to closed. If the service ends because of a failure event (exception and/or timeout), the circuit breaker changes the circuit state to open. While the circuit is in a half-open state and circuit breaker is already executing a request for the service, any other requests for the service wait until the circuit exits the half-open state. If service execution is successful, circuit breaker closes the circuit and executes the waiting requests for the service.

How Does a Circuit Breaker for a Service Work?

A circuit breaker works by monitoring a service for failures. The circuit breaker allows service execution to proceed when failure events do not meet an established threshold. However, when the number of failure events meets the established failure threshold within the failure time period, the circuit breaker opens the circuit, preventing further execution of the service.

The following table provides an overview of how a circuit breaker works.

Step	Description
1	The server receives a request to invoke a service. Upon receiving a request to invoke a service, the server first determines whether or not a circuit breaker is configured for the service. ■ If a circuit breaker is configured for the service, the server passes the request to the circuit breaker which checks the state of the circuit as described in step 2, below. ■ If a circuit breaker is not used with the service, the server invokes the service.
2	The circuit breaker examines the state of the circuit for the service. ■ If the circuit is closed, the circuit breaker invokes the service using a new thread from the circuit breaker thread pool, which is separate from the server thread pool. The thread that called the service originally waits for the results of the service execution. ■ If the circuit is open, the circuit breaker does not invoke the service. See step 5, below, for more information about how the circuit breaker responds to a request for service with an open circuit.

Step	Description
	■ If the circuit is half-open and this is the first request for the service since the circuit state became half-open, the circuit breaker invokes the service. When executing a service in a half-open state, any other requests for the service wait until the circuit exits the half-open state. For more information, see step 8, below.
3	When the circuit state is closed, upon execution of the service by the circuit breaker, one of the following occurs: ■ The service executes successfully. **Note:** If the **Failure event** property is set to **Timeout only** or **Exception or timeout**, a successful execution indicates that the service executed successfully within the specified timeout period. ■ The service ends with an exception. If the **Failure event** property is set to **Exception only** or **Exception or Timeout**, the circuit breaker considers the exception to be a failure event. The circuit breaker increments the failure count for the service by 1. ■ The service does not execute to completion before the timeout period elapses. If the **Failure event** property is set to **Timeout only** or **Exception or Timeout**, the circuit breaker considers the timeout to be a failure event. The circuit breaker increments the failure count for the service by 1. If the **Cancel thread on timeout** property is set to true, the circuit breaker attempts to cancel the thread executing the service. If the service ends with a failure event and it is the first failure event, the circuit breaker starts the failure timeout period. If the number of failure events specified in **Failure threshold** property occurs before the failure timeout period ends, the circuit opens. Circuit breaker determines the time that the failure period ends by adding the number of seconds specified in the **Failure period** property to the time the first failure event occurred. If the service executes successfully and completes after the failure timeout period ends, circuit breaker resets the failure count for the service to 0.
4	Subsequent invocations of the service result in the circuit breaker opening or “tripping” the circuit for the service if: ■ The number of failure events for the service equals the **Failure threshold** property value *and* ■ The failure events occur within the failure timeout period which circuit breaker determines by adding the value of **Failure period** property to the time of the first failure event.

Step	Description
	The circuit breaker starts the reset period which determines the length of time to keep the circuit in an open state. The **Circuit reset period** property determines the length of the reset period. For example, suppose that circuit breaker treats exceptions as a failure event, the **Failure threshold** property is set to 5, and the **Failure period** is set to 60 seconds. Further suppose that the first failure event occurs at 10:07:10. Circuit breaker starts the failure timeout period. The failure timeout period ends at 10:08:10 which circuit breaker determines by adding the **Failure period** value to the time of the first failure event. If 5 invocations of the service end with an exception in less than 60 seconds, that is, before 10:08:10, the circuit breaker opens the circuit for the service.
5	When the circuit breaker receives a request to execute the service, the circuit breaker does one of the following: ■ If the **Circuit open action** property is set to **Throw exception**, the circuit breaker responds to the invoke request by throwing the exception that caused the circuit to open. ■ If the **Circuit open action** property is set to **Invoke service**, the circuit breaker executes the alternate service specified in the **Circuit open service** property and returns the results to the calling service. The circuit breaker places the *$circuitBreakerService* and *$circuitBreakerEvent* parameters in the input pipeline for the alternate service. For more information about the circuit open service, see the section *Building a Service for Use with an Open Circuit Building a Service for Use with an Open Circuit* . **Note:** When the circuit is open, circuit breaker does not use the circuit breaker thread pool to throw the exception or execute the alternate, open circuit service. Instead, circuit breaker uses the same thread that executes the calling service to return the exception or execute the alternate, open circuit service.
6	The circuit breaker repeats the previous step in response to a request for the service until the time specified in the **Circuit reset period** property elapses.
7	When the circuit reset period elapses, the circuit breaker sets the circuit to a half-open state.
8	The circuit breaker receives a request for the service. ■ If this is the first request for the service since the circuit state became half-open, the circuit breaker invokes the requested service. One of the following occurs:

Step	Description
	■ If the service executes successfully, the circuit breaker closes the circuit. The circuit breaker will invoke the service upon subsequent service executions. The circuit breaker resets the failure count to 0 (zero). ■ If service execution results in a failure event, the circuit breaker re-opens the circuit and restarts the reset period. The circuit breaker proceeds as described in step 5.
	■ While the circuit is in a half-open state and circuit breaker is already executing a request for the service, any other requests for the service wait until the circuit exits the half-open state. If service execution is successful, circuit breaker closes the circuit and executes the waiting requests for the service. If service execution is not successful, circuit breaker re-opens the circuit and responds to the waiting services as described in step 5.

Configuring a Circuit Breaker for a Service

You can configure a circuit breaker for any user-defined service. Use the Service Development perspective in Software AG Designer to enable and configure a circuit breaker for a service. For configuration instructions, considerations, and guidelines, see the *webMethods Service Development Help*.

Building a Service for Use with an Open Circuit

When a circuit for a service is open, the service does not execute. Instead, circuit breaker responds to requests for the service by throwing the same exception that caused the last failure event or by executing an alternate service. The alternate service, known as the *open circuit service*, is a user-defined service that performs an action that makes sense for your application. In some applications, you might want the open circuit service to return a cached value or an alternate result to the calling service. You can also code the open circuit service to perform a different action based on whether it is the first invocation of the open circuit service since the circuit opened or if it is a subsequent invocation.

Whether a circuit breaker throws an exception or executes an alternate service depends on the value of the **Circuit open action** property. If you specify **Invoke service** as the circuit open action, you must identify the service to invoke in the **Circuit open service** property.When circuit breaker invokes the open circuit service, circuit breaker places parameters and values in the service pipeline for use by the open circuit service.

The following identifies and describes the parameters that circuit breaker places in the pipeline for an open circuit service.

Parameter Name	Description
$circuitBreakerService	**String.** Fully qualified name of the service with the open circuit.

Parameter Name	Description
$circuitBreakerEvent	**String.** Indicates whether this is the first time circuit breaker called the open circuit service since the circuit opened or if this is a subsequent invocation. The *$circuitBreakerEvent* parameter has one of the following values: ■ `CIRCUIT_OPENED` if this is the first execution of this open circuit service since the circuit opened. ■ `CIRCUIT_OPEN` if this is not the first execution of the open circuit service since the circuit opened.

Note:
The open circuit service has access to the parameters in the above table as well as any parameters that existed in the pipeline at the time of the execution request for the service with a configured circuit breaker.

Keep the following information in mind when building an open circuit service for use with a circuit breaker:

- If the open circuit service interacts with a remote resource, such as a database or web server, make the interaction asynchronous to prevent a service execution from blocking other threads or delaying the execution of the original calling service.
- An open circuit service can be used with more than one service with a configured circuit breaker.
- Circuit breaker does not use a thread from the circuit breaker thread pool to execute the open circuit service. Circuit breaker uses the same thread that executed the calling service to execute the open circuit service.
- If an exception occurs while executing this service, it does not impact the circuit breaker failure count for the originally requested service.
- Software AG recommends that the open circuit service is not configured to use a circuit breaker.
- Software AG recommends that the open circuit service is not the same as the service with the configured circuit breaker. That is, do not create a circular situation where a service with an open circuit calls itself.

Configuring the Circuit Breaker Thread Pool

Circuit breaker uses a dedicated thread pool, separate from the server thread pool, to execute services for which a circuit breaker is configured. This thread pool is referred to as the circuit breaker thread pool. You can specify the minimum and maximum number of threads in the circuit breaker thread pool.

At run time, circuit breaker uses a thread from the circuit breaker thread pool to execute the requested service, passing the service invocation pipeline to the new thread. This thread is separate from the thread executing the calling service. The calling service waits for a response from the requested service. Circuit breaker returns the service results and then returns the thread to the circuit breaker thread pool. The calling service then proceeds with execution. If the requested service is configured to treat a time out as a failure event and the service does not execute to completion before the timeout period elapses, circuit breaker returns an exception to the calling service. If the **Cancel thread on timeout** property is set to false, circuit breaker orphans the thread. If the **Cancel thread on timeout** property is set to true, circuit breaker attempts to cancels the thread. If the thread cannot be canceled, circuit breaker abandons the thread.

Circuit breaker uses a separate thread pool because it decouples the thread that executes the calling service from the thread that executes the requested service. This decoupling allows the calling service to proceed with execution if the requested service does not complete before the timeout period elapses. As a result, failures can return quickly. For example, suppose that a circuit breaker is configured for a service that reads information from a database. If the database goes off line, an attempt to connect to the unavailable database and execute a query may wait a while before returning because network input/output operations typically cannot be interrupted. By using a separate thread for the requested service, the circuit breaker can abandon the thread and return an exception to the client without needing to wait for the input/output operation to complete.

Note:
Circuit breaker uses the circuit breaker thread pool to execute only those services for which a circuit breaker is configured. Circuit breaker does not use a thread from the circuit breaker thread pool to execute the circuit open service.

You can configure the size of the circuit breaker thread pool by specifying the minimum and maximum number of threads for the pool. When the server starts, the circuit breaker thread pool initially contains the minimum number of threads. The server adds threads to the pool, as needed, until the pool contains the maximum number of allowed threads. If the pool reaches the maximum number of threads, before executing the next requested service with a configured circuit breaker, circuit breaker must wait for a thread to be returned to the pool.

The server provides server configuration parameters for specifying the minimum and maximum number of threads in the circuit breaker thread pool.

- watt.server.circuitBreaker.threadPoolMin specifies the minimum number of threads that the server maintains in the circuit breaker thread pool. The circuit breaker thread pool is used to execute services with a configured circuit breaker. When the server starts, the circuit breaker thread pool initially contains this minimum number of threads. The server adds threads to the pool as needed until it reaches the maximum allowed, which is specified by the watt.server.circuitBreaker.threadPoolMax. You must specify a value greater than or equal to 0 (zero) and less than or equal to the value of watt.server.circuitBreaker.threadPoolMax. The default is 10.

  Note:
  You must restart Microservices Runtime for changes to take effect.

- watt.server.circuitBreaker.threadPoolMax specifies the maximum number of threads that the server maintains in the circuit breaker thread pool. The circuit breaker thread pool is used to execute services with a configured circuit breaker. If this maximum number is reached, the server waits until services complete and return threads to the circuit breaker thread pool before running more services with a configured circuit breaker. You must specify a value greater than

0 and greater than or equal to the value of watt.server.circuitBreaker.threadPoolMin. The default is 75.

Note:
You must restart Microservices Runtime for changes to take effect.

Use the server configuration properties to size the circuit breaker thread pool appropriately for your environment. Keep in mind that all services for which a circuit breaker is configured share the circuit breaker thread pool.

Circuit Breaker Statistics

Microservices Runtime gathers circuit breaker statistics for each service with a configured circuit breaker. Microservices Runtime Administrator displays statistics in the **Circuit Breaker Information** table on the **Server > Service usage** page.

The following table identifies the circuit breaker information that Microservices Runtime maintains for each service with a configured circuit breaker.

Field	Description
Name	Name of the service for which a circuit breaker is configured.
State	The state of the circuit. The circuit state can be one of the following: ■ Closed. The service executes. ■ Open. The service does not execute. Instead, depending on the circuit breaker configuration, the service returns an exception or executes an alternative service. ■ Half-Open. The first request for this service since the circuit state changes to half-open results in service execution. All other requests wait. If the service executes successfully, the circuit is closed and waiting requests execute. If the service ends with a failure exception, the circuit is re-opened. For a detailed explanation of possible circuit states, see "Circuit States" on page 30.
Open Time	Time at which circuit breaker last set the circuit state to open.
Half-Open Time	Time at which circuit breaker last set the circuit state to half-open.
Closed Time	Time at which the circuit breaker last set the circuit state to closed.
Open Count	Number of times that circuit breaker set the circuit state to open since Microservices Runtime started.
Request Count	Number of incoming requests for the service since the circuit breaker changed the circuit state to open. For information about how circuit breaker handles

Field	Description
	requests for a service with an open circuit, see *How Does a Circuit Breaker for a Service Work? How Does a Circuit Breaker for a Service Work?* .

Note:
If a circuit breaker is not configured for any service, the **Circuit Breaker Information** table displays the message “No Services with a Circuit Breaker Enabled”.

5 Automatic Package Deployment

Automatic package deployment allows Microservices Runtime to install or upgrade packages automatically. When automatic package deployment is enabled, packages can be installed or upgraded without using Deployer or an administrator tool such as, Microservices Runtime Administrator. Automatic deployment can be used with an on-premises Microservices Runtime and an Microservices Runtime that runs inside a Docker container.

Automatically deploying packages can be particularly useful for a Microservices Runtime that runs in a Docker container. Instead of creating the Docker image containing individual Packages, creating a base image containing Default package can be used to bring in custom packages. Packages to be installed or updated on the container can be placed in an `autodeploy` directory that Microservices Runtime periodically scans. Microservices Runtime will find and install the new or updated packages. If you do not use automatic deployment, you must rebuild the Docker image to use any new or updated packages with Microservices Runtime container.

Note:
The automatic package deployment feature is available by default for a webMethods Microservices Runtime. To use the automatic package deployment feature with Integration Server, your Integration Server must have additional licensing.

When using the automatic package deployment with a Microservices Runtime that runs in a Docker container where the Microservices Runtime does not have web services functionality installed and the automatically deployed package contains web service assets (provider or consumer web service descriptors), one of the following occurs:

- If the automatically deployed package is a user-defined package, the package is partially loaded.
- If the automatically deployed package is a system package, the package is fully loaded.

However, the web service assets will be not work and Microservices Runtime writes a warning message to the server.log. For example: The Web service descriptor folder.subFolder:exampleWSD cannot be loaded. WebServices Support (WebServices) is not available in the installed product.

For more information about excluding web services from a Docker image created for Microservices Runtime, see *webMethods Integration Server Administrator's Guide* .

How Automatic Package Deployment Works

In automatic package deployment, the packages that you want Microservices Runtime to deploy automatically are placed in a location that Microservices Runtime periodically scans for new or updated custom packages. Microservices Runtime executes a system task named “Auto Package Deployer” that scans the folder and checks for new or updated custom packages. When Microservices Runtime finds new or updated custom packages, Microservices Runtime does one of the following:

- For a new package, Microservices Runtime installs, archives, and activates the package.
- For an updated custom package, Microservices Runtime does one of the following:
  - If hot deployment is enabled for automatic package deployment, Microservices Runtime uses the block-and-wait approach employed by hot deployment to deploy the package. This ensures that Microservices Runtime assets are available for processing without any

noticeable downtime. For more information about how hot deployment works, see *webMethods Integration Server Administrator's Guide*

- If hot deployment is not enabled, Microservices Runtime unloads the existing package information from memory and loads the new version of the package and its contents into memory. While the package is being upgraded, it is possible that some assets might be unavailable for some time. This could cause serious issues such as denial of service or disruption of the in-flight tasks.

Hot deployment is enabled for automatic package deployment if either of the following is true:

- Hot deployment is enabled globally for the Microservices Runtime. Hot deployment is enabled globally when the **Enabled** field is set to **Yes** on the **Settings > Hot deployment** page.
- Hot deployment is enabled for automatic package deployment when the watt.server.autodeploy.alwaysUseHotDeployment server configuration parameter is set to `true`.

By default, the automatic package deployment feature is disabled in Microservices Runtime. To enable automatic package deployment or configure other aspects of hot deployment, such as the system task execution frequency, see the section *Enabling and Configuring Automatic Package Deployment Enabling and Configuring Automatic Package Deployment* .

Determining Package Dependencies During Automatic Deployment

If you are deploying new package or updated custom packages and it has new dependent packages, you have to deploy the dependent packages too in the `autodeploy` folder. If you are deploying a new package or updated custom packages with existing dependent packages that are already deployed, then Microservices Runtime activates the new package. If you are deploying a new or updated custom packages with missing package dependencies the activation will fail. Also, if you are using hot deployment then you should identify package dependencies correctly. For more information see *webMethods Integration Server Administrator's Guide*.

Considerations for Auto Deployment of Packages

Before you configure automatic package deployment in Microservices Runtime, keep the following behavior limitations and considerations in mind.

- You cannot cancel an in-progress automatic package deployment operation.
- If the watt.server.autodeploy.alwaysUseHotDeployment parameter is set to `true`. Microservices Runtime performs hot deployment for the installation of any custom package regardless of whether it is a new package or an upgraded package. For more information on hot deployment, see "Enabling and Configuring Automatic Package Deployment" on page 42.
- If you are deploying new package and it has new dependent packages, you have to deploy all at once in the **autodeploy** folder.
- If you use hot deployment for automatic package deployment, all of the hot deployment considerations apply to automatic package deployment as well. For more information about

hot deployment considerations, see "Enabling and Configuring Automatic Package Deployment" on page 42.

- When hot deployment is used for automatic package deployment, Microservices Runtime uses the configured hot deployment values. For more information about configuring hot deployment, see "Enabling and Configuring Automatic Package Deployment" on page 42.

Enabling and Configuring Automatic Package Deployment

Microservices Runtime uses various configuration settings for automatic package deployment of new and updated packages. Most of the settings have defaults. You can change these settings on the **Settings > Extended settings** page of the Microservices Runtime Administrator as follows:

To configure automatic deployment for Microservices Runtime

1. In the Microservices Runtime Administrator, go to **Settings > Extended**.

2. Set the following server configuration properties in Microservices Runtime.

For this extended setting	Description
watt.server.autodeploy.enabled=true	Enables automatic package deployment .
watt.server.autodeploy.alwaysUseHotDeployment=true	Use hot deployment for automatic deployment of packages.
watt.server.autodeploy.interval=5	Specifies the interval, measured in minutes, at which Microservices Runtime executes the autodeploy system task.

3. Click **Save Changes**.

4. Restart the Microservices Runtime.

   The updated settings are now in effect.

Automatic Package Deployment Location

Where you place custom packages for automatic package deployment depends on whether you are using an on-premises Microservices Runtime or an Microservices Runtime running in a Docker container.

- For an on-premises Microservices Runtime, place packages that you want automatically deployed in this location:

  Integration Server_directory \replicate\autodeploy

Note:
For an on-premises Integration Server, place packages that you want automatically deployed in this location:

Integration Server_directory \instances*instance_name*\replicate\autodeploy

- For a Microservices Runtime or Integration Server running a Docker container, when you start the Docker container, you can map the above folder of a container to a volume which points to a folder on your HOST machine. You can then place packages that you want to the server to automatically deploy in the folder specified by the volume.

6 Using Configuration Variables Templates with Microservices Runtime

About Configuration Variables Templates

Microservices Runtime provides the ability to create a Docker image from an installed and configured instance of Microservices Runtime. The Docker image contains the Microservices Runtime application including packages and Microservices Runtime assets such as ports, JMS connection aliases, keystores, and JDBC pools. Because you can execute multiple Docker containers from a single Docker image, a Docker image for a Microservices Runtime instance might be used across development, testing, and production environments. While the microservices and accompanying artifacts are the same across the running containers and stages, configuration information might be different. For example, user name and password combinations, proxy server host and port, remote server host and port, and global variable values might be different across different instances of running Docker containers. While a Microservices Runtime stores configuration information inside its file system and therefore inside any Docker image created from the Microservices Runtime, some configuration information can be externalized and passed to the Microservices Runtime at startup. With Microservices Runtime you can accomplish this through use of a configuration variables template and environment variables.

A configuration variables template contains configuration properties that map to properties on the Microservices Runtime. The property values can be set externally in the template and then passed to a Microservices Runtime when it starts up. As part of the startup process, Microservices Runtime loads the information from the configuration variables template and creates or replaces the configuration information stored in the file system.

By externalizing configuration information, a single Docker image created for a Microservices Runtime can be used across multiple environments, including different stages of the production cycle. For example, you might use different templates for specific environments such as testing versus production. Or you might use the same template for all environments but use environment variables to vary the configuration in each environment.

While the primary purpose of configuration variables is to extend the usefulness of single Microservices Runtime image by making it possible to reuse a single Docker image across multiple stages in the production cycle, you can also use configuration variables templates in situations where other deployment options are too heavyweight.

Note:
The configuration variables feature is available by default for Microservices Runtime. To use the configuration variables feature with Integration Server, your Integration Server must have additional licensing.

What Does a Configuration Variables Template Look Like?

A configuration variables template is a properties file that contains configuration data as a series of key-value pairs where the key name reflects the asset and particular asset property for which you can supply a value. For example, `jndi.DEFAULT_IS_JNDI_PROVIDER.providerURL=nsp://myHost:9000` indicates that the JNDI provider alias named DEFAULT_IS_JNDI_PROVIDER has a URL of nsp://myHost:9000.

When a configuration variables template is generated from an existing Microservices Runtime, the template is populated with configuration data from the Microservices Runtime instance. The template content reflects the Microservices Runtime configuration at the time Microservices Runtime generated the template. For example, the configuration variables template lists a key-value pair for each defined global variable.

The generated configuration variables template does not contain key-value pairs for all of configuration information for Microservices Runtime. This is because only a subset of configuration information is supported for use in configuration variables template.

Below is an excerpt of a configuration variables template:

```
#Sample Generated Template
#Wed Jun 20 17:13:40 EDT 2018
email.WmRoot.myEmailPort.host=exchange
email.WmRoot.myEmailPort.password=******
email.WmRoot.myEmailPort.server_port=7891
email.WmRoot.myEmailPort.user=user123ispasswordexpirationsettings.
jms.DEFAULT_IS_JMS_CONNECTION.clientID=DEFAULT_IS_JMS_CLIENT
jndi.DEFAULT_IS_JNDI_PROVIDER.providerURL=nsp\://localhost\:9000
settings.watt.server.compile=C\:\\IS_10-3\\June6\\jvm\\jvm\\bin\\javac -
classpath {0} -d {1} {2}
settings.watt.ssh.jsch.ciphers=aes256-ctr,aes192-
ctr,arcfour,arcfour128,arcfour256,aes128-ctr,aes128-cbc,3des-ctr,3des-
cbc,blowfish-cbc,aes192-cbc,aes256-cbc
```

Only specific values can be supplied via the template for an asset. For example, for a JMS connection alias a configuration variables template lists a key-value pair for the clientID, user, and password properties. The template omits key-value pairs for all other properties.

In the template, each property name follows this pattern: `assetType.assetName.propertyName=value` where `assetType` is the type of administrative asset, `assetName` is the name given to the asset such as an alias name, and `propertyName` is the asset property for which you can set a value.

For example, following are key-value pairs for a global variable named `serverhost`, a global variable named `serverport`, the DEFAULT_IS_JMS_CONNECTION connection alias, and the JDBC pool alias named webMPool:

```
globalvariable.serverhost.value=server123.example.com
globalvariable.serverport.value=5555
jms.DEFAULT_IS_JMS_CONNECTION.clientID=DEFAULT_IS_JMS_CLIENT
jms.DEFAULT_IS_JMS_CONNECTION.password=******
jms.DEFAULT_IS_JMS_CONNECTION.user= userA
jdbc.webMPool.dbURL=jdbc:wm:oracle://testserver:1521;serviceName=webm
jdbc.webMPool.password={AES}CszC/Yl1w1XqEK0B17RFucP0kgMHNt823RUU6ad39tw\=
jdbc.webMPool.userid=jdbcuserTest
```

To change the assigned values in a configuration variables template, use a text editor to open the template and make changes. Any value you specify for a property must adhere to the requirements for that property. For example, the global variable value cannot exceed 255 characters. For passwords or other types of values that should be encrypted, Microservices Runtime Administrator provides a utility to encrypt a value. Microservices Runtime Administrator uses password handles and the Password-Based Encryption technology installed with Microservices Runtime for encryption.

You can also add an asset by editing the configuration variables template. Not all assets for which you can edit configuration in a configuration variables template can be created via a template.

As an alternative to "hard coding" a value for a property in a template, you can specify an environment variable (ENV variable) as the value of a property in the configuration variables template. Using ENV variables can make the configuration variables template more flexible. For example, the following line specifies that the JDBC pool URL for the alias webMPool should be set to the value of the environment variable named `JDBC_URL`: `jdbc.webMPool.dbURL=$env{JDBC_URL}`

If you opt to use an ENV variable as a value for configuration variables, make sure that you specify the values for ENV variables defined in the configuration variables template. Refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers.

When Is the Template Applied?

Microservices Runtime applies a configuration variables template at startup. When Microservices Runtime starts, either in a Docker container or on-premises, Microservices Runtime looks for a configuration variables template in the following locations in the specified order.

1. The location identified by the `SAG_IS_CONFIG_PROPERTIES` environment variable.
2. *Integration Server_directory* /instances/*instanceName*/application.properties
3. *Integration Server_directory* /application.properties

Once Microservices Runtime locates a template, Microservices Runtime does not look for a template in any remaining locations.

Note:
If Microservices Runtime does not find a configuration variables template at any of the above locations, Microservices Runtime proceeds with start up and does not apply a template.

Microservices Runtime processes the template, updating the configuration information in Microservices Runtime with the information in the template. While processing the template, Microservices Runtime writes log messages to the logs/configurationvariables.log to reflect the progress of the variable substitution. When updating is complete, Microservices Runtime proceeds with startup.

After applying a fix with PIE-73984 (IS_10.11_Core_Fix4 and higher), if Microservices Runtime cannot apply an application.properties file because the file is malformed, Microservices Runtime logs errors about the malformed file to the server log and error log.Microservices Runtime writes the following to the error log: `Dynamic variables processor has encountered an error. Please refer to configuraitonvariables.log for details.` Microservices Runtime logs the actual exception to the error log. Prior to application of IS_10.11_Core_Fix4, Microservices Runtime does not start if a malformed application.properties file was supplied.

If the Microservices Runtime license file cannot be read during start up and a configuration variables file is present, Microservices Runtime does not proceed with start up and logs the following error:

```
[ISS.0028.9999C] com.softwareag.is.dynamicvariables.DynamicVariablesCriticalException:
The license provided is invalid or cannot be read. Remove application.properties file
or provide a valid MSR license
```

When updating the configuration, if Microservices Runtime encounters an asset that does not exist, one of the following happens:

- If the asset is one of the types that can be created through a configuration variables template, Microservices Runtime creates it using the supplied key-value pairs. Only a subset of all the properties for an asset can be set in a configuration variables template. Microservices Runtime uses default values for all other asset properties. For a list of assets that can be created via application of a configuration variables templates, see "Creating New Assets with the Configurations Variable Template" on page 59.
- If an asset type cannot be created through a configuration variables template, Microservices Runtime ignores properties and values that do not correspond to existing assets on the Microservices Runtime.

During substitution or asset creation, Microservices Runtime sets values for properties that are specified in the template but not set on the Microservices Runtime. For example, suppose the configuration variables template file specifies the following for a JMS connection alias named `myAlias`:

```
jms.myAlias.clientID=abc
jms.myAlias.password=encryptedValue
jms.myAlias.user= userA
```

If the Microservices Runtime contains a JMS connection alias named "myAlias" and that alias specifies a **Connection Client ID** value of "xyz", at startup, Microservices Runtime substitutes "abc" for "xyz". If the myAlias connection alias does not specify values for **Username** or **Password**, Microservices Runtime substitutes the values set in `jms.myAlias.password` and `jms.myAlias.user` in the application properties template for the JMS connection alias **Username** and **Password** fields.

Approaches for Using a Configuration Variables Template with Microservices Runtime

Because a configuration variables template can be used with a Microservices Runtime running in a Docker container or an on-premises Microservices Runtime, there are multiple approaches to using the template. The approach you select determines the template file name, the template file location, whether the template file is part of a Docker image, and how to start the Microservices Runtime so that Microservices Runtime applies the template.

To help determine which approach to use, decide the following:

- Will the Microservices Runtime with which you want to use the template run in a Docker container or on-premises (outside of a Docker container)?
- If Microservices Runtime runs in a Docker container, do you want to include the template as part of the Docker image or external to the image?
- Do you want to hard code values for variables in the template or use environment (ENV) variables to specify some or all values?

The answers to the above questions help determine which approach to use, which in turn, determines the files that you need to create, where you place the configuration variables template, the name of the file, and how to start the Microservices Runtime so that Microservices Runtime applies the template values.

The following table describes the possible approaches for using a configuration variables template with Microservices Runtime

Approach	Description and requirements
Docker image does not include template which does not use ENV variables	Microservices Runtime runs in a Docker container, the template is not included in the Docker image, and configuration variables values are hard coded in the template only. The template does not use ENV variables. Template file name: application.properties or a user-defined name Template location: A location that is accessible to the Docker container. Start up: The `docker run` command includes the `SAG_IS_CONFIG_PROPERTIES` environment variable which specifies the name and location of the configuration variables template.
Docker image does not include template and template uses ENV variables	Microservices Runtime runs in a Docker container, the template is not included in the Docker image, and configuration variable values are hard coded in the template and/or the template uses environment variables. Template file name: application.properties or a user-defined name Template location: A location that is accessible to the Docker container. If you are using ENV variables for some or all of the variable values, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers. Start up: The `docker run` command includes the `SAG_IS_CONFIG_PROPERTIES` environment variable which specifies the name and location of the configuration variables template. If using the `docker run` command, the command must also include options `--env, -e` or `--env-file` options for setting the ENV variables in the container.
Docker image includes template which uses ENV variables	Microservices Runtime runs in a Docker container, the configuration variables template is part of the Docker image, and the template uses environment variables for some or all the values in the template. Template name: application.properties Template location is one of the following: ■ *Integration Server_directory* /instances/*instanceName*/application.properties ■ *Integration Server_directory* /application.properties

Approach	Description and requirements
	If you are using ENV variables for some or all of the variable values, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers. Start up: The `docker run` command must include options `--env`, `-e` or `--env-file` options for setting the ENV variables in the container.
On-premises Microservices Runtime installation does not include template and template does not use ENV variables.	Microservices Runtime runs on-premises, configuration variables values are hard coded in the template only, and the template is not included in the Microservices Runtime file system. The template does not use ENV variables. Template name: application.properties or a user-defined name Template location: A location that is accessible to the Microservices Runtime. Start up: Microservices Runtime start up includes the `SAG_IS_CONFIG_PROPERTIES` ENV variable which specifies the name and location of the configuration variables template.
On-premises Microservices Runtime includes template which does not use ENV variables.	Microservices Runtime runs on-premises and a configuration variables template defines all the values. The template does not use ENV variables. Template name: application.properties Template location is one of the following: ■ *Integration Server_directory* /instances/*instanceName*/application.properties ■ *Integration Server_directory* /application.properties Start up: This approach does not require any special steps at start up.
On-premises Microservices Runtime includes template which uses ENV variables	Microservices Runtime runs on-premises and a configuration variables template hard codes the values and/or uses environment variables for the values. Template name: application.properties Template location is one of the following: ■ *Integration Server_directory* /instances/*instanceName*/application.properties ■ *Integration Server_directory* /application.properties

Approach	Description and requirements
	Environment variables can be specified in *Integration Server_directory* /bin/setenv.bat(sh) files or ENV variables can be pre-defined in your environment. For an Integration Server with an Microservices Runtime license, environment variables can be specified in *Integration Server_directory* profiles/IS_*instanceName*/bin/custom_setenv.bat(sh) files or ENV variables can be pre-defined in your environment. Start up: This approach does not require any special steps at start up.

Overview of Building a Configuration Variables Template

Preparing a configuration variables template for use with Microservices Runtime consists of a series of general tasks. The order in which you complete these tasks may be affected by whether you intend to use the template with a Microservices Runtime in a Docker container or an on-premises Microservices Runtime.

The following table identifies the general tasks that you complete for preparing a configuration variables template for use with Microservices Runtime.

Task	Description
1	Decide how you want to use the configuration variables template and how you want to supply values. Do you want to use the template with a Microservices Runtime running in a Docker container or an on-premises Microservices Runtime? Do you want to supply all the variable values in the template or do you want to use an environment variables as well? For more information, see “Approaches for Using a Configuration Variables Template with Microservices Runtime ” on page 49.
2	Configure a Microservices Runtime such that the generated template will contain the properties that you want to control. When generating a template, the Microservices Runtime includes properties only for those assets which exist on the Microservices Runtime.
3	Generate the configuration variables template for the Microservices Runtime. For more information about creating a configuration variables template, see “Generating a Configuration Variables Template” on page 53.
4	Edit the configuration variables template to contain the property values that you want the Microservices Runtime to modify at startup. For more information about this stage, see “Editing a Configuration Variables Template” on page 55. If you want to add assets to Microservices Runtime by applying configuration variables template, add the key-value pairs for the asset and its properties to the template. Keep in mind that you can use a configuration variables template to add only certain asset types to the Microservices Runtime. For more information about adding assets, see “Creating New Assets with the Configurations Variable Template” on page 59.

Task	Description
5	Place the configuration variables template in the correct location with the appropriate name. The template location depends on whether or not you are using the template with a Docker image or on-premises Microservices Runtime and if the template is part of the image or file system. For more information, see "Approaches for Using a Configuration Variables Template with Microservices Runtime " on page 49. If you are using ENV variables as values in the configuration variables template, refer to the Docker documentation or the documentation for your container orchestration tool for more information about how to specify the ENV variables for Docker containers.
6	If you are using the template in a Docker container and the template resides inside the Microservices Runtime file system, build the Docker image. For more information about creating a Docker image, see *webMethods Integration Server Administrator's Guide webMethods Integration Server Administrator's Guide* For more information about creating a Docker image, see the *webMethods Integration Server Administrator's Guide*.

Generating a Configuration Variables Template

Microservices Runtime provides a utility to generate a configuration variables template based on the configuration of a running Microservices Runtime. This configuration variables template reflects the current configuration of the Microservices Runtime.

After the Microservices Runtime is configured, you can use the utility to generate the template. The resulting template serves as the starting point for creating a customized configuration variables template.

Before you generate the configuration variables template from a Microservices Runtime, you might want to:

- Make sure to favorite (star) all of the server configuration parameters that you want to set with the template on the **Settings > Extended** page.
- Ensure that the keystores and truststores are loaded. When a keystore or truststore is loaded it indicates that the keystore or truststore configuration is valid and that the Microservices Runtime initialized the keystore or truststore alias successfully. To see if a keystore or truststore is loaded, navigate to the **Security > Keystore** page and ensure that the **Loaded** column displays **Yes**.

Note:
You can create a configuration variables template manually. You do not need to generate a template from an existing configured Microservices Runtime. However, generating the template from a Microservices Runtime is a time-saving option that provides a starting point. A generated template may also be less prone to errors with key names in the template than a template created manually.

To generate a configuration variables template

1. Open the Microservices Runtime Administrator for the Microservices Runtime from which you want to generate the variables

2. Go to **Microservices > Configuration variables**.

3. Click **Generate Configuration Variables Template**.

   Microservices Runtime generates an application.properties file. The application.propeties file includes the text `#Sample Generated Template` followed by the date and time of the generation.

4. Download the application.properties file and save it to your preferred location for editing.

The generated application.properties file includes the following:

- The date and time that Microservices Runtime generated the template.
- Key-value pairs for asset properties for which a value is specified on the Microservices Runtime. The template does not include variables for assets that do not exist or for asset properties that are not specified.

  For a list of assets and properties for which Microservices Runtime generates configuration variables see "Configuration Variables Template Assets" on page 85.

  Note:
  In the template, any key that contain a period (.) as part of its name is escaped using another period. For example, for a JMS connection alias named `my.JMS.alias`, the template property corresponding to the Client ID in the alias the name: `jms.my..JMS..alias.clientID=value`

- Key-value pairs for the server configuration parameters that were configured to show on the Extended Settings page only.
- Keystore and truststore alias properties for keystores and truststores that were loaded at the time of template generation only.
- The text ****** for any passwords that appear as property values in the template.
- The environment variable name in the format `$env[`*`environmentVariableName`*`]` for any key for which an environment variable was used as the value.
- The Kubernetes variable itself in the format `$secret[`*`secretName`*`]` for any key for which a Kubernetes Secret was used as the value.
- For a configured asset, the configuration variables template lists properties only for which a value is specified. For example, if you specify a client ID for a JMS connection alias, but do not specify a user name or password for use with the JMS connection alias, the configuration variables template contains `jms.`*`aliasName`*`.clientID=`*`value`* but not `jms.`*`aliasName`*`.password` or `jms.`*`aliasName`*`.user`.

Editing a Configuration Variables Template

After you generate a configuration variables template from a Microservices Runtime, you modify the configuration variables template to contain the property values that you want applied to a Microservices Runtime at startup. You can hard code values or specify an environment variable as the value. You can also encrypt values or use a Kubernetes Secret to protect sensitive data. In addition, you can also add assets to the template.

Keep the following information in mind when you edit a configuration variables template.

- Any value that you specify must meet the requirements for the associated property. For example, global variable values cannot exceed 255 characters. If the value is not valid, failures will occur at runtime, either during server initialization or when clients start sending in requests.
- A configuration variables template can be used to change the configuration information for assets that already exist on the Microservices Runtime.
- You can add assets for some asset types to a Microservices Runtime by adding key-value pairs to a template. Any asset that you create must meet the naming requirements for the asset. If it does not, the improperly configured asset might result in Microservices Runtime failure at start up or during operations. For a list of assets that you can add, "Overview of Building a Configuration Variables Template" on page 52
- The configuration variables template lists only assets that existed on the Microservices Runtime from which you generated the template. You can add an asset for an asset type for which asset creation is allowed via a configuration variables template. If asset creation is not allowed for that type but you know the asset will exist on the Microservices Runtime that will use the template, you can add the asset and the necessary key-value pairs to the template. At startup, if the template specifies a value for a non-existent asset, the Microservices Runtime logs a message to the configuration variables log.

  For more information about assets and properties and the corresponding key name, see "Configuration Variables Template Assets" on page 85.
- When Microservices Runtime generates an application.properties template, any properties that contain an alias name with a period (.) include another period as an escape character. For example, for a JMS connection alias named `my.JMS.alias`, the template property corresponding to the Client ID in the alias the name: `jms.my..JMS..Alias.clientID=value`

  Do not remove the period (.) acting as the escape character.

  If you edit the template by adding a key-value pair for an asset property and the asset property includes an alias name with a period (.), make sure to add another period to serve as the escape character.

To supply a value in the configuration variables template

1. Open the template in a text editor.
2. For each key for which you want to supply a value, do one of the following:

To	Enter this for the property value
Hard code a value	The value you want to use.
Use an environment value as the value	The name of the environment variable in the format: $env{*environmentVariableName*} Where *environmentVariableName* is the name of the environment variable. For example: `$env{JDBC_URL}`
Use a Kubernetes Secret as the value	`$secret{SecretName}` Where `secretName` is the key for the secret whose value you want to use. For example: `$secret{mySecretPassword}` For more information about using a Kubernetes Secret with a configuration variables template, see "Using a Kubernetes Secret with a Configuration Variables Template" on page 58.
Encrypt a value	Use Microservices Runtime Administrator to generate an encrypted value and then copy that value into the template. For more information see,"Encrypting Values for the Variables Template" on page 57. While Microservices Runtime accepts clear text passwords as well as encrypted ones, Software AG recommends that you encrypt all passwords and other sensitive data in your configuration variables templates.

3. Repeat the above step for each key that you want to set.

4. To add an asset, add the key-value pair for each property of the asset that you want to set.

   For example, to add a global variable, insert the following into the template:

   `globalvariable.globalVariableName.value`

   Where *globalVariableName* is the name of the global variable that you want to create.

5. Repeat step 4 for every asset that you want to create.

6. Save the template.

Encrypting Values for the Variables Template

You may want your configuration variables template to use encrypted values for sensitive data such as passwords. Microservices Runtime provides a way to generate an encrypted value which can then be placed in the template.

For encryption, Microservices Runtime uses AES to encrypt the values for configuration variables, in particular AES/ECB/PKCS5Padding. When a Docker image is created for Microservices Runtime using the Docker script is_container.bat/sh, the script bundles a secret key with the image, allowing the encrypted values from one instance to be decrypted and reused in another.

Instead of encrypting values, you can use Kubernetes Secrets for confidential data such as passwords. For more information about Kubernetes Secrets, see "Using a Kubernetes Secret with a Configuration Variables Template" on page 58.

Note:
When you generate a configuration variables template, Microservices Runtime replaces any passwords and any secret global variables with ****** .

Software AG recommends that you encrypt all passwords and other sensitive data in your configuration variables templates.

To encrypt a value

1. Open the Microservices Runtime Administrator for the Microservices Runtime that you want to use to generate an encrypted value.

2. Go to **Microservices > Configuration variables**.

3. Click **Generate Encrypted Configuration Variables**.

4. In the **Value** field, enter the value that you want encrypted.

   By default, Microservices Runtime Administrator masks any characters that you enter in the **Value** field. Click the **Show Value** check box if you want Microservices Runtime Administrator to display the characters in clear text.

5. Click **Encrypt**.

   Microservices Runtime encrypts the value and displays the encrypted value in the **Encrypted Value** field.

6. Click **Copy** to copy the encrypted value to the clipboard.

7. Open the configuration variables template to which you want to add the encrypted value, locate the key for which you want to use the value, and then paste the copied value into the template.

8. Repeat steps 4 –7 for each value you want to encrypt.

Using a Kubernetes Secret with a Configuration Variables Template

A Kubernetes Secret is an object that contains a small piece of confidential data such as credentials. The Secret is separate from a Docker container image, meaning the confidential data can be externalized. You can use a Kubernetes Secret with Microservices Runtime by specifying the key for the Secret in the configuration variables file that is passed into a Microservices Runtime running in a Docker container.

Microservices Runtime supports one Secret per container (each Secret can contain multiple entries often called keys) and a Secret of type Opaque only.

Prior to using a Secret with a configuration variables files (application.properties), you must do the following:

- Create the Secret with Kubernetes
- Mount the Secret as a data volume

To use a Kubernetes Secret for a property value in a configuration variables file, set the key-value pair for the asset property using the following format:

```
key=$secret{SecretName}
```

Where the `SecretName` is the key for the secret whose value you want to use. For example:

```
truststore.DEFAULT_JVM_TRUSTSTORE.ksPassword=$secret{TruststorePasswordSecretName}
```

```
user.userName.password=$secret{mySecretPassword}
```

When you start the container, you can set the `SECRET_PATH` environment variable to the path where the Secret will be present inside the container. You do not need to specify this environment variable if the Secret is mounted at /etc/secrets.

When Microservices Runtime applies the configuration variables file, Microservices Runtime looks for the location of the Secret by first checking if the `SECRET_PATH` environment variable is present. If so, then Microservices Runtime reads the Secret from the path present in this environment variable. If the `SECRET_PATH` environment variable was not provided when the container started, then Microservices Runtime expects the Secret will be mounted at /etc/secrets and will read the Secret from there.

If Microservices Runtime cannot find the Secret used in the configuration variables file (the Secret is not present or access to the file is restricted due to file permissions), the following Info level message will be written to the configurationvariables.log:

[ISS.0028.0039I] Exception occurred while reading secret file: <*pathToFile*>

Creating New Assets with the Configurations Variable Template

For some asset types, you can add assets to Microservices Runtime by adding key-value pairs to the configuration variables template that is applied at start up. Any asset that you create must meet the naming requirements for the asset. An incorrectly configured asset can cause Microservices Runtime start up fail or result in failure during later server operations.

To add an asset to a configuration variables template, follow the procedure and guidelines in "Editing a Configuration Variables Template" on page 55.

The following list identifies the asset types for which you can create an asset through application of a configuration variables template:

- Broker connection alias

  Note:
  Only one Broker connection alias can exist on a Microservices Runtime (or an Integration Server). If the application.properties file being applied to the Microservices Runtime specifies a Broker connection alias, but a Broker connection alias already exists on Microservices Runtime, an error is written to the configurationvariables.log and the Broker configuration is not updated.

- Consul
- Email port
- File polling port
- Global variable
- JDBC pool alias
- JMS connection alias
- JNDI provider alias
- Keystore alias
- LDAP configuration
- Proxy server alias
- Remote server alias
- SFTP server alias
- SFTP user alias
- Truststore alias
- Universal Messaging connection alias (webMethods messaging)
- User account

Template File Locations

When the configuration variables template is ready for use with a Microservices Runtime, you need to save the template to a place accessible to Microservices Runtime upon startup. Where you place the configuration variables template file and any accompanying environment variables list depends on whether you are using the template with Microservices Runtime running in a Docker container or on-premises.

You can place the template file in one of the following locations.

- The location to be identified by the SAG_IS_CONFIG_PROPERTIES environment variable.
- *Integration Server_directory* /instances/*instanceName*

  For this option, the configuration variables template must be a file named application.properties. If you use this location with Microservices Runtime running in a Docker container, create the Docker image after completing the template and placing it here.
- *Integration Server_directory*

  For this option, the configuration variables template must be a file named application.properties. If you use this location with Microservices Runtime running in a Docker container, create the Docker image after completing the template and placing it here.

For more details about where to place the configuration variables template and an environment variables list, if you are using one, see "Approaches for Using a Configuration Variables Template with Microservices Runtime " on page 49.

Providing a Configuration Variables Template when Starting a Docker Container

When running a Microservices Runtime image in a Docker container, you can specify the configuration variables template and/or environment (ENV) variables in the `docker run` command.

Example

The following `docker run` command uses the `SAG_IS_CONFIG_PROPERTIES` ENV variable to specify the name and location of the configuration variables template. In this example, the Docker image for the Microservices Runtime is named is:microPkg and exposes ports 5555 and 9999. Additionally, the customApplication.properties file location is accessible by the Docker container using Docker volume.

```
docker run -d --name IS_Default -p 5555 -p 9999
-v
/opt/softwareag/customApplication.properties:/opt/softwareag/customApplication.properties

-e SAG_IS_CONFIG_PROPERTIES=/opt/softwareag/customApplication.properties
is:microPkg
```

Example

The following `docker run` command uses the `SAG_IS_CONFIG_PROPERTIES` ENV variable to specify the name and location of the configuration variables template and uses the `--env-file` option to specify the list of ENV variables that are used in the configuration variables template. In this example, the Docker image for the Microservices Runtime is named is:microPkg and exposes ports 5555 and 9999.

```
docker run -d --name IS_Default -p 5555 -p 9999
-e SAG_IS_CONFIG_PROPERTIES=/opt/softwareag/customApplication.properties
-v
/opt/softwareag/customApplication.properties:/opt/softwareag/customApplication.properties

--env-file env.list is:microPkg
```

Configuration Variables Logging

When Microservices Runtime applies the configuration variables template at startup, Microservices Runtime has not yet initialized the journal logger which is used for the server log. For this reason, Microservices Runtime uses a separate logging capability for logging Microservices Runtime messages related to processing a configuration variables template. The configuration variables log contains messages about the operations, warnings, and errors that occur while Microservices Runtime applies the template.

Below is an excerpt from the configuration variables log:

```
2018-06-18 15:34:10 EDT [ISS.0028.0016I] Configuration variables template file
found at C:\SoftwareAG\IntegrationServer\application.properties will be used for
processing.
2018-06-18 15:34:12 EDT [ISS.0028.0007I] Changed property in
C:\SoftwareAG\IntegrationServer\instances\default\config\server.cnf. Old value:
localhost. New value: localhost,127.0.0.1. Configuration variable key:
watt.net.proxySkipList.
2018-06-18 15:34:12 EDT [ISS.0028.0017W] Encryptor is creating password handle
because the given password handle
wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https does not exist in
the password store.
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0017W] Encryptor is creating password handle
because the given password handle
wm.is.admin.WSEndpoint.transport.CONSUMER.HTTPS.ws_cons_https does not exist in the
password store.
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.transport.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0008I] Set password in secure password store.
Password handle: wm.is.admin.WSEndpoint.message.CONSUMER.HTTPS.ws_cons_https
2018-06-18 15:34:12 EDT [ISS.0028.0007I] Changed property in
C:\SoftwareAG\IntegrationServer\instances\default\config\kerberos.cnf. Old
value: . New value:
```

Each log message is prefixed with the time stamp and a message ID that includes the logging facility (0028), message code, and the severity level. The severity levels are described in the following table:

	Level	Description	Example
C	Critical	Could not execute the configuration variables processing or an error occurred that caused Microservices Runtime initialization to stop.	Configuration file `C:\SoftwareAG\IntegrationServer\instances\default\packages\WmPublic\config\listeners.cnf` not found
E	Error	One or more of the intended configuration changes could not be applied	`java.io.FileNotFoundException:` `C:\SoftwareAG\IntegrationServer\instances\default\config\jndi\jndi_BasicAuth.properties` (The system cannot find the file specified)
W	Warning	A configuration variable was applied, but there is something about which you should be aware.	Property corresponding to key username not found in XML file proxy.cnf.
I	Information	Informational messages about the progress of configuration variables processing.	Changed property in `C:\SoftwareAG\IntegrationServer\instances\default\config\server.cnf.` Old value: localhost New value: localhost,127.0.0.1 Configuration variable key: watt.net.proxySkipList.
D	Debug	Low level messages, helpful for troubleshooting.	Processor found for the assetType truststore.

Microservices Runtime does not support filtering the contents of the configuration variables log based on logging level. By default, the configuration variables log includes all messages with a severity of Information or higher. To include debug messages in the log, you must set the `SAG_IS_CONFIG_VARIABLES_DEBUG` environment variable to true at startup.

Microservices Runtime writes the configuration variables log messages to the console (STDOUT) and/or to the location: *Integration Server_directory* /instances/*instanceName*/logs/configurationvariables.log. A Microservices Runtime running in a Docker container always writes the configuration variables log messages to both locations.

Note: Microservices Runtime running in a Docker container also writes the server log to the console. Configuration variables log messages include “ISS.0028” in the message ID which distinguishes the messages from server log messages.

When starting an on-premises Microservices Runtime from the command line, you can specify the destination for the configuration variables log using the `-log` switch.

The following table identifies the command line options for specifying the destination of the configuration variables log file.

Command Line Option	Log Destination
`-log none`	The console (STDOUT).
`-log both`	The console (STDOUT) and *Integration Server_directory* /instances/*instanceName*/logs/configurationvariables.log.

Note:
If the `-log` switch is not specified, Microservices Runtime writes the log messages to this location only: *Integration Server_directory* /instances/*instanceName*/logs/configurationvariables.log

Starting Microservices Runtime from the command line is the same as starting Integration Server from the command line.

For information about starting Integration Server from the command line, see *webMethods Integration Server Administrator's Guide*.

Microservices Runtime does not preserve the contents of the configurations variable log across restarts. When running on-premises Microservices Runtime overwrites the existing configurations variable log upon startup.

Viewing the Applied Template for a Microservices Runtime

For a running Microservices Runtime you may want to see the configuration variables templates that Microservices Runtime applied at startup. This can help you compare current configuration to configuration at the time of startup. Using Microservices Runtime Administrator, you can obtain the application.properties file used at startup. If the application.properties used environment variables as values, Microservices Runtime substitutes the actual value for the environment variable value.

To view the applied configuration variables template

1. Open Microservices Runtime Administrator for which you want to view the applied template.
2. Go to **Microservices > Configuration variables**.
3. Click **Get Active Configuration Variables Template**.

   Microservices Runtime generates an application.properties file that contains the key-value pairs from the applied temple. The application.properties file includes the text `#No Active Template` if Microservices Runtime did not apply a template at startup.

7 Monitoring Microservices Runtime

Overview of Monitoring Microservices Runtime

Microservices Runtime provides capabilities for monitoring the health of a Microservices Runtime and gathering metrics about the server and the microservices it contains. External applications such as container management and monitoring tools, can use the metrics and data supplied by Microservices Runtime to help determine whether the Microservices Runtime is performing correctly or optimally.

Microservices Runtime exposes the monitoring features via endpoints. Requests sent to the endpoint result in the invocation of an internal service that gathers data and then returns the status and/or payload do the requester.

Microservices Runtime includes the following monitoring capabilities:

- Health gauge which returns an overall up or down status for a Microservices Runtime based on a set of health indicators. The health gauge endpoint is http://*host*:*port*/health
- Metrics which returns server and service metrics in Prometheus configuration format. The metrics endpoint is http://*host*:*port*/metrics

Note:
The monitoring features are available by default for Microservices Runtime. To use the monitoring features with Integration Server, your Integration Server must have additional licensing.

About the Health Gauge

The health gauge returns an overall UP or DOWN status for the Microservices Runtime based on the collective status of enabled health indicators. When the `health` endpoint is invoked, Microservices Runtime executes all of the enabled health indicators. A health indicator determines the UP or DOWN status of a specific component of Microservices Runtime. If all of the health indicators return an UP status, the entire Microservices Runtime is considered to be up. The health gauge returns a HTTP 200 status code to the requester. If even one of the health indicators returns a DOWN status, the entire Microservices Runtime is considered to be down. The health gauge returns a HTTP 503 status code to the requester.

Regardless of the overall status, the response includes a payload in JSON format that contains more details for each health indicator, including whether the indicator returned a status of UP or DOWN. By default, the response follows the ASCII order. For example, the ServiceThread health indicator returns the current number of service threads in use, the current number of available threads, and the maximum number of available threads.

Predefined Health Indicators

Microservices Runtime includes predefined health indicators for some of the basic components of a Microservices Runtime. Some, but not all, of the health indicators have a configurable property that you can use to specify the threshold at which a health indicator returns an UP or DOWN status.

The following table describes the predefined health indicators included with Microservices Runtime.

Indicator Name	Description
Cluster	Checks for the number of available servers in a cluster of Microservices Runtime servers. Returns a status of UP if the number of available servers is greater than or equal to the defined minimum in the **Number of cluster hosts** property for the Cluster health indicator. Otherwise, returns a status of DOWN. This health indicator is returned only for a Microservices Runtime that is a member of a configured cluster. That is the Microservices Runtime must be a member of a stateful cluster.
Diskspace	Checks for low disk space. Returns status of UP if the percentage of free disk space is greater than what is specified in the **Free disk space threshold** property for the Diskspace health indicator. Otherwise, returns a status of DOWN.
HybridConnections	Checks all the listed tenant connection aliases and associated accounts for webMethods Cloud. Returns a status of UP if all of the listed tenant connections and associated accounts are UP. Returns a status of DOWN if any of the listed tenant or account connections are down. The health indicator includes individual statuses for tenant connections and associated accounts, along with the overall hybrid connection status. If an enabled tenant does not have an account alias or the account alias is not enabled, the health indicator does not list the tenant alias. If a tenant is disabled and the associated account is enabled, the health indicator shows the account status as UP. This behavior is as designed. **Note:** The hybrid connectivity alerts are introduced as part of PIE-81171 in IS_10.11_Core_Fix9.
JDBC	Checks for available JDBC connections across all JDBC functions such as ISInternal and ISCoreAudit. Returns a status of UP if, for each JDBC connection pool, Microservices Runtime can obtain a valid JDBC connection before a 200 millisecond time out elapses. Otherwise, returns a status of DOWN. The timeout value is not configurable. The JDBC health indicator skips any JDBC functions that do not have an associated pool.
JMS	Checks that JMS connection alias are available. Returns a status of UP if all enabled JMS connection aliases are active, meaning that Microservices Runtime can ping the JMS Provider or create a connection successfully. Otherwise, returns a status of DOWN.
JNDIAliases	Checks that the connections for a JNDI aliases are up by attempting to make a connection for JNDI.

Indicator Name	Description
Memory	Checks for low available memory. Returns a status of UP if the percentage of free memory is greater than what is specified for the **Free memory threshold** property for the Memory health indicator. Otherwise, returns a status of DOWN.
RemoteServers	Checks the status of remote servers. Returns a status of UP if Microservices Runtime can successfully invoke the internal service wm.server:ping on each server for which there exists a remote server alias. Otherwise, returns a status of DOWN.
SFTPServers	Checks the connection to remote SFTP serves for which an SFTP server alias is configured. Returns a status of UP if a connection can be obtained for all of SFTP server aliases with at least one SFTP user alias.
ServiceThread	Checks for low available server threads. Returns a status of UP if the percentage of available server threads is greater than what is specified in the **Available threads threshold** property for the ServiceThread health indicator. Otherwise, returns a status of DOWN.
Sessions	Checks for low available licensed sessions. Returns a status of UP if the percentage of available licensed sessions is greater than the value specified for the **Used licenses threshold** property for the Sessions health indicator. Otherwise, returns a status of DOWN.
UMAliases	Checks that the Universal Messaging connection aliases for webMethods messaging. Returns a status of UP if all of the enabled Universal Messaging connection aliases are available. Otherwise, returns a status of DOWN.

Note:
Products installed on top of Microservices Runtime might provide their own health indicators.

Enabling and Disabling Health Indicators

Whether or not a health indicator is enabled determines if the healthy gauge includes the indicator when determining the UP or DOWN status of the Microservices Runtime. If you do not want the health gauge to include a particular indicator when determining the overall UP or DOWN status, disable the indicator. A disable indicator does not execute when the health endpoint is invoked.

To enable or disable a health indicator

1. Go to **Microservices > Health gauge**.

2. In the Health Indicator List, do one of the following:

   - To enable a disabled health indicator, click **No** in the **Enabled** column.
   - To disable an enabled health indicator, click **Yes** in the **Enabled** column.

Health Indicator Properties

Some of the health indicators have a configurable property that determines when a health indicator returns a status of UP or DOWN. For example, the ServiceThreads health indicator has the **Available threads threshold** which specifies the percentage of the server threads that must be available for the indicator to return a status of UP. You can edit the threshold to tailor the indicator to your environment.

The following table identifies the configurable properties for the predefined health indicators.

Health Indicator	Property Name	Value
Cluster	**Number of cluster hosts**	Specify the minimum number of cluster members that must be available for the Cluster health indicator to return a status of UP. When the number of servers in the cluster is less than the specified minimum number, the Cluster health indicator returns a status of DOWN. The default is 2.
Disks-pace	**Free disk space threshold (as percentage of maximum available disk space)**	Specify the percentage of free disk space out of the maximum available disk space above which the Diskspace health indicator returns a status of UP. When free disk space on the host or container on which Microservices Runtime resides is less than or equal to the specified percentage, the Diskspace health indicator returns a status of DOWN. The default is 10 percent.
Memory	**Free memory threshold (as percentage of maximum memory)**	Specify the percentage of free memory above which the Memory health indicator returns a status of UP. When free JVM memory for Microservices Runtime is less than or equal to the specified percentage, the Memory health indicator returns a status of DOWN. The default is 10 percent.
ServiceThread	**Available threads threshold (as percentage of maximum server threads)**	Specify the percentage of available server threads in the server thread pool at which the ServiceThread health indicator returns a status of UP. When the percentage of available threads is less than or equal to the specified percentage, the ServiceThread health indicator returns a status of DOWN. The default is 10 percent.
Sessions	**Used licenses threshold (as percentage of total licensed sessions)**	Specify the percentage of used licensed sessions at which the Sessions health indicator returns a status of DOWN. When the percentage of available licensed sessions is less than or equal to the specified percentage, the Sessions health indicator returns a status of DOWN. The default is 85 percent.

Configuring Health Indicator Properties

You can edit the properties of a health indicator to tailor the indicator for your environment. A health indicator with one or more configurable properties appears as a hypertext link in the Health Indicators List on the **Microservices > Health Gauge** page.

To configure health indicator properties

1. Go to **Microservices > Health gauge**.
2. In the Health Indicator List, click the name of the health indicator you want to configure.
3. On the ***IndicatorName* Properties** page, click **Edit** next to the property name.
4. In the Value field, set a new threshold value for the property.
5. Click **Save Changes**.

Invoking the Health Gauge

You can invoke the health gauge via the `health` endpoint on the Microservices Runtime. When Microservices Runtime runs in a Docker container, you can use the `health` endpoint to monitor the state of the container from tools such as Kubernetes.

The request URL for the `health` endpoint is:

```
http://<hostname>:<port>/health
```

Where *<hostname>* is the IP address or name of the machine and *<port>* is the port number where Microservices Runtime is running.

For a Microservices Runtime or an Integration Server running in a Docker container that was created using the is_container.bat/sh script, the default ACL for accessing the `healths` endpoint is “Anonymous”, which means authentication is not required when the endpoint is invoked. For all other Microservices Runtimes or an Integration Servers, the default ACL is “Administrator”, which means that authentication is required to access the endpoint. However, you can use the environment variable SAG_IS_HEALTH_ENDPOINT_ACL to set the ACL whose members can invoke the `health` endpoint. For more information about environment variables, see *webMethods Integration Server Administrator’s Guide webMethods Integration Server Administrator’s Guide* .

Note:
The `health` endpoint is a predefined URL alias named “health” for the internal service that executes all enabled health indicators. Software AG does not recommend editing the predefined “health” URL alias. If you migrate to Microservices Runtime version 10.3 or higher from an earlier version and you already have a URL alias named “health”, Microservices Runtime does not create a health URL alias that points to the internal service. Any invocations of the `health` endpoint will not result in execution of health indicators. If you want to use the health gauge and the associated health indicators, you need to rename your existing health URL alias. Upon restart, Microservices Runtime creates a new health URL alias that corresponds to the `health` endpoint.

Obtaining Metrics for a Microservices Runtime

Microservices Runtime can generate metrics about the server and services on the server that the Prometheus server can use to provide insight to the operation of the Microservices Runtime and the services it contains. Microservices Runtime generates metrics in a Prometheus format. Prometheus is an open source monitoring and alerting toolkit which is frequently used for monitoring microservices.

Microservices Runtime exposes the metrics generating feature via the `metrics` endpoint. When the `metrics` endpoint is invoked, Microservices Runtime gathers server, service-level, and JVM metrics and returns the data in a Prometheus format.

Microservices Runtime can also gather and return metrics for when the `metrics` endpoint is called.

For a detailed list of the metrics returned by Microservices Runtime, see "Prometheus Metrics Returned by Microservices Runtime " on page 102.

Note:
The Microservices Runtime documentation assumes a familiarity with Prometheus technology. An in-depth discussion of Prometheus is beyond the scope of this guide but is available elsewhere.

Invoking the Metrics Endpoint

To instruct Microservices Runtime to gather metrics, you invoke the `metrics` endpoint on the Microservices Runtime. The request URL would be:

```
http://<hostname>:<port>/metrics
```

Where *`<hostname>`* is the IP address or name of the machine and *`<port>`* is the port number where Microservices Runtime is running.

Invocation of the `metrics` endpoint is restricted to users with Administrator access.

For a Microservices Runtime or an Integration Server running in a Docker container that was created using the is_container.bat/sh script, the default ACL for accessing the `metrics` endpoint is "Anonymous", which means authentication is not required when the endpoint is invoked. For all other Microservices Runtimes or an Integration Servers, the default ACL is "Administrator", which means that authentication is required to access the endpoint. However, you can use the environment variable SAG_IS_METRICS_ENDPOINT_ACL to set the ACL whose members can invoke the `metrics` endpoint. For more information about environment variables, see *webMethods Integration Server Administrator's Guide webMethods Integration Server Administrator's Guide* .

Note:
The `metrics` endpoint is a predefined URL alias named "metrics" for the internal service that gathers statistics. Software AG does not recommend editing the predefined "metrics" URL alias. If you migrate to Microservices Runtime version 10.3 or higher from an earlier version and you already have a URL alias named "metrics", Microservices Runtime does not create a metrics URL alias that points to the internal service. Any invocations of the `metrics` endpoint will not result in the gathering and return of metrics. If you want to use the metrics gathering

functionality, you need to rename your existing metrics URL alias name. Upon restart, Microservices Runtime creates a new metrics URL alias that corresponds to the `metrics` endpoint.

8 Consul Support

Configuring Connections to Consul Server

Configure one or more server aliases for the public services provided in WmConsul to use to connect to a Consul server. You must create at least one server alias for the services to execute successfully. An alias name used as the *registryAlias* input parameter value for a WmConsul public service must exist in the Microservice Consul Registry Server alias list.

Note:
The WmConsul package contains the public services for interacting with a Consul server.

To create an alias to the Consul server

1. Open Microservices Runtime Administrator.
2. Go to **Packages > Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, click **Create Microservice Consul Registry Server Alias**.
5. Under Microservice Consul Registry Server Alias Properties, provide the following information.

For this field	Specify
Alias	Name for the server alias.
Host Name or IP Address	Location of the Consul server as a host name or IP address.
Port Number	Port on which to connect to the Consul server.
Enable Consul Health Check	Whether to enable a health check for microservices registered using this alias. Consul uses the health check to keep track of the health of a registered microservice . Select **Yes** to enable health checks.
User Name	Optional. If the Consul server is configured to use a user name and password, the user name.
Password	Optional. If the Consul server is configured to use a user name and password, the password.
Use SSL	Whether to use a secure connection to connect to the Consul server. To use SSL, select **Yes** and then provide truststore and possibly keystore information. **Note:**

For this field	Specify
	If you select yes, the Consul server must be configured to accept HTTPS requests.
Keystore Alias	Optional. Keystore alias that contains the client certificates to use for the secure connection. You need to provide this information only if **Use SSL** is set to Yes and the registry server requires client certificates.
Key Alias	Optional. Key alias for the private key and certificates to use for establishing a secure connection. You need to provide this information only if **Use SSL** is set to **Yes** and the Consul server requires client certificates .
Truststore Alias	Optional. Truststore alias that contains the Consul server's certificate authority certificates. You need to provide this information only if **Use SSL** is set to Yes.

6. Click **Save Changes**.

Testing an Alias for the Consul Server

After you add an alias for a Consul server, you can test the alias to ensure that Microservices Runtime can establish a connection to the Consul server using the information provided in the alias.

To test a Consul server alias

1. Open Microservices Runtime Administrator.
2. Go to **Packages > Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, in the Microservce Consul Registry Server List click ▶ in the Test column for the alias you want to test.

   Microservices Runtime Administrator displays a status message above the list of aliases that indicates whether or not the connection is successful.

Setting the Default Alias for the Consul Server

You can identify one of the Consul server aliases as the default alias. The pub.consul.client services will use this alias to connect to the Consul server if you do not specify a different alias in the *registryAlias* input parameter.

To specify the default Consul server alias

1. Open Microservices Runtime Administrator.
2. Go to **Packages > Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, click **Change Default Alias**.
5. On the Microservice Consul Registry Servers > Change Default Alias page, in the **Default Alias** list, select the Consul server alias to use as the default.
6. Click **Update**.

Deleting a Consul Server Alias

If you no longer need an alias to a Consul server, you can delete it.

To delete a Consul server alias

1. Open Microservices Runtime Administrator.
2. Go to **Packages > Management**.
3. Click the icon for the WmConsul package.
4. On the Microservice Consul Registry Servers page, in the Microservice Consul Registry Server List click ✕ in the **Delete** column for the alias you want to delete. Microservices Runtime Administrator prompts you to confirm deleting the alias. Click **OK**.

   If you delete the default Consul server alias, Microservices Runtime Administrator displays a status message stating there is no longer a default Consul server alias which can prevent microservice registration.

Consul Public Services Folder

The following table identifies the elements available in the consul folder of the WmConsul package:

Element	Package and Description
	WmConsul. De-registers a microservice from a Consul server. Use as a shutdown service for the package being registered as a microservice.

Element	Package and Description
	WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul.
	WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul and if there are multiple hosts for this microservice, randomly return one of those hosts.
	WmConsul. Registers a microservice with a Consul server. Use as a startup service for the package being registered as a microservice.

pub.consul.client:deregisterService

WmConsul. De-registers a microservice from a Consul server. Use this service as a shutdown service for the package being registered as a microservice.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice name to de-register from the Consul server.

Output Parameters

None

Usage Notes

A package that identifies pub.consul.client:deregisterService as a shutdown service must identify WmConsul as a package dependency.

pub.consul.client:getAllHostsForService

WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice for which you want to retrieve a list of active hosts.

Output Parameters

hostList	String List. An array containing the names of hosts of the microservice. Each element is in the format *host*:*port* (for example, appserver.xyz.com:4555). If there are no hosts registered for the given microservice, returns null.

pub.consul.client:getAnyHostForService

WmConsul. Queries the Consul server for a list of active hosts that have registered the given microservice with Consul and if there are multiple hosts for this microservice, randomly return one of those hosts.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias
microserviceName	String. Microservice for which you want to retrieve an active host.

Output Parameters

hostName	String. Name of a host of the microservice, chosen randomly. The host name is in the format: *port* (for example, appserver.xyz.com:4555). If there are no hosts registered for the given microservice, returns null.

pub.consul.client:registerService

WmConsul. Registers a microservice with a Consul server. Use this service as a startup service for the package being registered as a microservice.

Input Parameters

registryAlias	String. Optional. Alias to use to connect to a Consul server. If you do not specify this parameter, the service uses the default Consul server alias.

microserviceName	String. Microservice name to register with the Consul server.

Output Parameters

None

Usage Notes

A package that identifies pub.consul.client:registerService as a startup service must identify WmConsul as a package dependency.

A Microservices Runtime vs Integration Server

Microservices Runtime vs Integration Server Feature Comparison

Microservices Runtime is a superset of Integration Server. In general, this means that Microservices Runtime includes features that are not part of a standard Integration Server. However, some of the features available on a standard Integration Server are not installed by default on Microservices Runtime and must be selected specifically during the installation of Microservices Runtime. Additionally, because Microservices Runtime is optimized for running in a Docker container, Microservices Runtime does not include all of the behavior available on an Integration Server, such as running multiple instances on the same host machine or the use of Command Central. However an Integration Server equipped with a Microservices license provides all of the functionality of Integration Server and Microservices Runtime without the size and speed optimizations made for a Microservices Runtime.

The following table compares features and functionality available across Integration Server, Integration Server equipped with Microservices license, and Microservices Runtime.

Feature	**Integration Server**	**Integration Server with a Microservices license**	**Microservices Runtime**
Integration Server - core functionality	Yes	Yes	Yes
OSGi platform	Yes	Yes	No
Command Central support	Yes	Yes	No
Common Directory Services Support	Yes	Yes	Optional
Java Service Wrapper developed by Tanuki Software, Ltd.	Yes	Yes	No
Multi-instance support	Yes	Yes	No
Running as a Windows service	Yes	Yes	No
Digital Event Services support	Yes	Yes	No
External RDMS support	Yes	Yes	Optional
Publishing to CentraSite	Yes	Yes	Optional
Consul registry (WmConsul package)	No	No	Yes
WmParquet package	Yes	Yes	No
Circuit breaker support	No	Yes	Yes
Configuration variables support	No	Yes	Yes

Feature	Integration Server	Integration Server with a Microservices license	Microservices Runtime
Automatic package deployment	No	Yes	Yes
Health check endpoint	No	Yes	Yes
Metrics endpoint with metrics in Prometheus format	No	Yes	Yes
CloudStreams	Yes	Yes	Yes
Deployer	Yes	Yes	Yes

Note:
When a Docker image is built for a Microservices Runtime the web service functionality can be excluded. This reduces the size of Docker image. Any web service descriptors that reside on the Microservices Runtime will not execute. Additionally, attempts to execute any of the following web service related built-in services results in a ServiceException:

- pub.soap* services
- pub.utils.ws.setCompatibilityModeFalse
- pub.client:soapClient
- pub.client:soapHTTP
- pub.client:soapRPC
- pub.trigger:createJMSTrigger, when *jmsTriggerType*=`SOAPJMS`

For more information about excluding web services from a Docker image created for Microservices Runtime, see *webMethods Integration Server Administrator's Guide* .

B Configuration Variables Template Assets

A configuration variables template is a properties file that contains configuration data as a series of key-value pairs where the key name reflects the asset and particular asset property for which you can supply a value. A configuration variables template contains key-value pairs for a subset of Microservices Runtime configuration only. That is, a configuration variables template does not include key-value pairs for all of the configuration information for a Microservices Runtime. Only some configuration information is supported for use with configuration variables templates. Assets can be modified and created when Microservices Runtime applies the template at start up.

The following table identifies the assets and property names that correspond to the key names in a template. An asterisk (*) next to an asset indicates that the asset can be created through application of a configuration variables template.

Asset	Property	Key Name
Broker connection alias*	**Broker Host**	`messaging.BROKERALIAS.brokerHost`
	Broker Name	`messaging.BROKERALIAS.brokerName`
	Client Group	`messaging.BROKERALIAS.clientGroupName`
	Keystore	`messaging.BROKERALIAS.certfile`
	Keystore Password	`messaging.BROKERALIAS.keystorePassword`
	Truststore	`messaging.BROKERALIAS.truststore`
	Username	`messaging.BROKERALIAS.user`
	Password	`messaging.BROKERALIAS.password`
Cache Manager	**Terracotta Server Array URLs**	`cachemanager.`*`cacheManagerName`*`.urls`
Consul*	**Host Name or IP Address**	`consul.`*`aliasName`*`.host`
	Password	`consul.`*`aliasName`*`.password`
	Port Number	`consul.`*`aliasName`*`.port`

Asset	Property	Key Name
	User Name	`consul.`*`aliasName`*`.user`
Dashboard configuration	**Indexer thread count**	`statisticsdatacollector.monitorConfig.luceneMemorySize`
	Maximum JDBC pool connections	`statisticsdatacollector.monitorConfig.maxJdbcPoolConnections`
	Maximum number of results	`statisticsdatacollector.monitorConfig.maxResults`
	Memory size (MB)	`statisticsdatacollector.monitorConfig.luceneMemorySize`
	Retention days	`statisticsdatacollector.monitorConfig.retentionDay`
	Searcher thread count	`statisticsdatacollector.monitorConfig.searcherThreadCount`
	Statistics Data Collection	`statisticsdatacollector.monitorConfig.enabled=true`
Email Port*	**Host Name**	`email.`*`packageName`*`.`*`aliasName`*`.host`
	Password	`email.`*`packageName`*`.`*`aliasName`*`.password`
	Port	`email.`*`packageName`*`.`*`aliasName`*`.server_port`
	Type	`email.`*`packageName`*`.`*`aliasName`*`.type`
	User Name	`email.`*`packageName`*`.`*`aliasName`*`.user`
File Polling Port*	**Completion Directory**	`filepolling.`*`packageName`*`.`*`aliasName`*`. completionDir`
	Error Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`. errorDir`
	Monitoring Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`. monitorDir`
	Working Directory	`filepolling.`*`packageName`*`.`*`aliasName`*`. workDir`
Health Indicators	**Enabled**	`healthindicators.Diskspace.enabled`
	Free disk space threshold	`healthindicators.Diskspace.properties.threshold.value`
	Enabled	`healthindicators.JDBC.enabled`
	Enabled	`healthindicators.JMS.enabled`
	Enabled	`healthindicators.JNDIAliases.enabled`
	Enabled	`healthindicators.Memory.enabled`

Asset	Property	Key Name
	Free memory threshold	`healthindicators.Memory.properties.threshold.value`
	Enabled	`healthindicators.RemoteServers.enabled`
	Enabled	`healthindicators.SFTPServers.enabled`
	Enabled	`healthindicators.ServiceThread.enabled`
	Available threads threshold	`healthindicators.ServiceThread.properties.threshold.value`
	Enabled	`healthindicators.Sessions.enabled`
	Used licenses threshold	`healthindicators.Sessions.properties.threshold.value`
	Enabled	`healthindicators.UMAliases.enabled`
Global Variable*	**Value**	`globalvariable.`*`globalVariableName`*`.value`
JDBC Functional Alias	**Associated Pool Alias**	`jdbcfunc.`*`functionName`*`.connPoolAlias` **Note:** The specified JDBC connection pool alias must exist before the JDBC functional alias is updated.
JDBC Pool Alias*	**Database URL**	`jdbc.`*`aliasName`*`.dbURL`
	Associated Driver Alias	`jdbc.`*`aliasName`*`.driverAlias`
	User ID	`jdbc.`*`aliasName`*`.userid`
	Password	`jdbc.`*`aliasName`*`.password`
JMS Connection Alias*	**Enabled**	`jms.`*`aliasName`*`.enabled` Set to `true` to enable. Set to `false` to disable.
	Description	`jms.`*`aliasName`*`.description`
	Transaction Type	`jms.`*`aliasName`*`.transactionType` Set to: ■ `0` for NO_TRANSACTION. ■ `1` for LOCAL_TRANSACTION. ■ `2` for XA_TRANSACTION.
	Connection Client ID	`jms.`*`aliasName`*`.clientID`

Asset	Property	Key Name
	User	`jms.`*`aliasName`*`.user`
	Password	`jms.`*`aliasName`*`.password`
	Create Connection Using	`jms.`*`aliasName`*`.associationType` Set to: ■ `0` for JNDI LOOKUP. ■ `1` for NATIVE WEBMETHODS API.
	JNDI Provider Alias Name	`jms.`*`aliasName`*`.jndi_jndiAliasName` The specified JNDI alias must match an existing JNDI provider alias.
	Connection Factory Lookup Name	`jms.`*`aliasName`*`.jndi_connectionFactoryLookupName`
	Create Administered Objects On Demand (Universal Messaging)	`jms.`*`aliasName`*`.jndi_automaticallyCreateUMAdminObjects` Set to `true` to create a connection factory or destination on the JNDI provider if the object is not found at the time Integration Server looks up the object. Otherwise, set to `false`.
	Enable Follow the Master (Universal Messaging)	`jms.`*`aliasName`*`.jndi_enableFollowTheMaster` Set to `true` to follow the master realm server. Set to `false` to disable following the master realm server.
	Monitor webMethods Connection Factory	`jms.`*`aliasName`*`.jndi_connectionFactoryUpdateType` Set to: ■ `NO` - Do not monitor the webMethods Connection Factory. ■ `CLIENT_POLL` - Poll for changes at a specified interval. ■ `JNDI_POLL` - Poll for changes at the interval defined by webMethods Connection Factory object. ■ `NOTIFICATION` - Monitor the connection factory by registering an event listener.
	Polling Interval (minutes)	`jms.`*`aliasName`*`.jndi_connectionFactoryPollingInterval`
	Broker Host	`jms.`*`aliasName`*`.nwm_brokerHost`

Asset	Property	Key Name
	Broker Name	`jms.`*`aliasName`*`.nwm_brokerName`
	Client Group	`jms.`*`aliasName`*`.nwm_clientGroup`
	Broker List (optional)	`jms.`*`aliasName`*`.jms.jmsAliasName.nwm_brokerList`
	Keystore (optional)	`jms.`*`aliasName`*`.nwm_keystore`
	Keystore Type (optional)	`jms.`*`aliasName`*`.nwm_keystoreType` Set to `JKS` or `PCKS12`.
	Truststore (optional)	`jms.`*`aliasName`*`.nwm_truststore`
	Truststore Type (optional)	`jms.`*`aliasName`*`.nwm_truststoreType` Set to `JKS`.
	Class Loader	`jms.`*`aliasName`*`.classLoader` Set to: ■ `INTEGRATION_SERVER` ■ `PACKAGE:` *`PackageName`* where *PackageName* is the name of the package class loader.
	Maximum CSQ Size	`jms.`*`aliasName`*`.csqSize`
	Drain CSQ in Order	`jms.`*`aliasName`*`.csqDrainInOrder` Set to `true` to drain in order. Set to `false` to disable draining in publishing order.
	Create Temporary Queue	`jms.`*`aliasName`*`.optTempQueueCreate` Set to `true` to create a temporary queue for request-reply operations that do not specify a replyTo destination. Otherwise, set to `false`.
	Enable Request-Reply Listener for Temporary Queue	`jms.`*`aliasName`*`.allowReplyToConsumer` Set to `true` to create a single dedicated MessageConsumer for receiving synchronous replies delivered to the temporary queue for this JMS connection alias. Set to `false` to create a new JMS MessageConsumer for each reply message.
	Enable Destination Management with Designer (Broker and	`jms.`*`aliasName`*`.manageDestinations` Set to `true` to allow creating destinations on the messaging provider using Designer. Otherwise, set to `false`.

Asset	Property	Key Name
	Universal Messaging)	
	Create New Connection per Trigger	`jms.`*`aliasName`*`.allowNewConnectionPerTrigger` Set to `true` to create a separate connection per concurrent JMS trigger. Otherwise, set to `false`.
	Caching Mode	`jms.`*`aliasName`*`.producerCachingMode` Set to: ■ `0` to disable. ■ `2` to enable per destination.
	Minimum Pool Size (unspecified destinations)	`jms.`*`aliasName`*`.producerSessionPoolMinSize`
	Maximum Pool Size (unspecified destinations)	`jms.`*`aliasName`*`.producerSessionPoolSize`
	Minimum Pool Size Per Destination	`jms.`*`aliasName`*`.cacheProducersPoolMinSize`
	Maximum Pool Size Per Destination	`jms.`*`aliasName`*`.cacheProducersPoolSize`
	Destination Lookup Names (semicolon delimited)	`jms.`*`aliasName`*`.cacheProducersQueueList`
	Topic List (semicolon delimited)	`jms.`*`aliasName`*`.cacheProducersTopicList`
	Idle Timeout (milliseconds)	`jms.`*`aliasName`*`.poolTimeout`
	Max Retry Attempts	`jms.`*`aliasName`*`.producerMaxRetryAttempts`
	Retry Interval (milliseconds)	`jms.`*`aliasName`*`.producerRetryInterval`
	Logging Type	`jms.`*`aliasName`*`.um_loggingOutput` Set to: ■ `0` for SERVER LOG.

Asset	Property	Key Name
		■ 1 for MESSAGING AUDIT LOG.
	Enable Producer Message ID Tracking	`jms.`*`aliasName`*`.um_producerMessageTracking` Set to `true` to enable, `false` to disable.
	Producer Message ID Tracking: Include Destinations (semicolon delimited)	`jms.`*`aliasName`*`.um_producerIncludedStrings`
	Enable Consumer Message ID Tracking	`jms.`*`aliasName`*`.um_consumerMessageTracking` Set to `true` to enable, `false` to disable.
	Consumer Message ID Tracking: Include Triggers (semicolon delimited)	`jms.`*`aliasName`*`.um_consumerIncludedStrings`
JNDI Provider Alias*	**Initial Context Factory**	`jndi.`*`aliasName`*`.initialContextFactory`
	Provider URL	`jndi.`*`aliasName`*`.providerURL`
	Provider URL Failover List	`jndi.`*`aliasName`*`.providerURLFailoverList`
	Security Principal	`jndi.`*`aliasName`*`.securityPrincipal`
	Security Credentials	`jndi.`*`aliasName`*`.securityCredentials`
Kerberos	**Realm**	`kerberos.KerberosConfig.kerberosConfig`
	Key Distribution Center Host	`kerberos.KerberosConfig.kerberosKDC`
	Kerberos Configuration File	`kerberos.KerberosConfig.kerberosRealm`
	Use Subject Credentials Only	`kerberos.KerberosConfig.useSubjectCreds`
Keystore Alias*	**Description**	`keystore.`*`aliasName`*`.ksDescription` **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Type	`keystore.`*`aliasName`*`.ksType`

Asset	Property	Key Name
		Default supported values are: `JKS` and `PKCS12`. Other keystore types may be available by loading additional security providers, or by setting the `watt.security.keyStoreTypes` server configuration property. **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Provider	`keystore.`*`aliasName`*`.ksStoreProviderName` Default supported values are as follows: - If `keystore.`*`aliasName`*`.ksType` is set to `JKS`: `SUN` - `keystore.`*`aliasName`*`.ksType` is set to `PCKS12`: `BC`, `SUN`, or `SunJSSE` **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Location	`keystore.`*`aliasName`*`.ksLocation`
	Password	`keystore.`*`aliasName`*`.ksPassword`
	Key Alias Password	`keystore.`*`aliasName`*`.keyAlias.`*`keyAliasName`*`.keyAliasPassword`
	HSM Based Keystore	`keystore.`*`aliasName`*`.ksIsHsm` Set to `true` to specify that the keystore is stored on a Hardware Security Module (HSM) device. If set to true, the value of the `keystore.`*`aliasName`*`.ksLocation` is ignored. **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
LDAP Configuration*	**Directory URL**	`ldap.`*`ldapDirectory`*`.url`
	Principal	`ldap.`*`ldapDirectory`*`.prin`
	Credentials	`ldap.`*`ldapDirectory`*`.password`
	Synthesize DN and **Query DN**	`ldap.`*`uuid`*`.useaf`

Asset	Property	Key Name
		Set to true to specify that Microservices Runtime builds a distinguished name by adding a prefix and suffix to the username. A value of true is equivalent to selecting the **Synthesize DN** option on the **Security > User Management > LDAP configuration** page. Set to false to specify that Microservices Runtime builds a query that searches a specified root directory for the user. A value of false is equivalent to selecting the **Query DN** option on the **Security > User Management > LDAP configuration** page. The default is true.
	UID Property	`ldap.`*`uuid`*`.uidprop`
	Group Member Attribute	`ldap.`*`uuid`*`.mattr`
	User Root DN	`ldap.`*`uuid`*`.userrootdn`
	Group ID Property	`ldap.`*`uuid`*`.gidprop`
	Group Root ID	`ldap.`*`uuid`*`.grouprootdn`
	Connection Timeout	`ldap.`*`uuid`*`.timeout`
	Minimum Connection Pool Size	`ldap.`*`uuid`*`.poolmin`
	Maximum Connection Pool Size	`ldap.`*`uuid`*`.poolmax`
	DN Prefix	`ldap.`*`uuid`*`.dnprefix`
	DN Suffix	`ldap.`*`uuid`*`.dnsuffix`
	User Email	`ldap.`*`uuid`*`.useremail`
	Default Group	`ldap.`*`uuid`*`.group`
	The *uuid* is a unique identifier for the LDAP directory. When an application.properties file is generated by Microservices Runtime, the UUID is automatically generated by Microservices Runtime. When using an application.properties file to create an LDAP configuration, make sure to specify a unique value in the UUID portion of the key name.	
Proxy Server Alias*	**Host Name or IP Address**	`proxyserver.`*`aliasName`*`.host`

Asset	Property	Key Name
	Port Number	`proxyserver.`*`aliasName`*`.port`
	User Name	`proxyserver.`*`aliasName`*`.username`
	Password	`proxyserver.`*`aliasName`*`.password`
Proxy Server Bypass	**Addresses**	`settings.watt.net.proxySkipList`
Remote Server Alias*	**Host Name or IP Address**	`remoteserver.`*`aliasName`*`.host`
	Port Number	`remoteserver.`*`aliasName`*`.port`
	User Name	`remoteserver.`*`aliasName`*`.user`
	Password	`remoteserver.`*`aliasName`*`.password`
	Retry Server	`remoteserver.`*`aliasName`*`.retryServer`
Server configuration parameters (server.cnf)	Any	`settings.`*`serverConfigurationParameterName`*
SFTP Server Alias*	**SFTP Client Version**	`sftpserver.`*`aliasName`*`.version` The SFTP client to use. Set to v1 or v2 .
	Host Name or IP Address	`sftpserver.`*`aliasName`*`.hostName` The hostname or IP address of the SFTP server.
	Port Number	`sftpserver.`*`aliasName`*`.port` The port number of the SFTP server.
	Proxy Alias	`sftpserver.`*`aliasName`*`.proxyAlias` The proxy alias through which requests are routed.
	Host Key	`sftpserver.`*`aliasName`*`.hostKey` The public key of the SFTP server.
	Min DH Key Size	`sftpserver.`*`aliasName`*`.minDHKeySize` The minimum DH key size. Not applicable to SFTP Client Version 1.
	Max DH Key Size	`sftpserver.`*`aliasName`*`.maxDHKeySize`

Asset	Property	Key Name
		The maximum DH key size. Not applicable to SFTP Client Version 1.
	Preferred Key Exchange Algorithms	`sftpserver.`*`aliasName`*`.preferredKeyExchangeAlgorithm` Preferred key exchange algorithms separated by a comma.
	Preferred MAC Algorithms S2C	`sftpserver.`*`aliasName`*`.preferredMACS2C` Message Authentication Code (MAC) Server to Client algorithms separated by a comma.
	Preferred MAC Algorithms C2S	`sftpserver.`*`aliasName`*`.preferredMACC2S` Message Authentication Code (MAC) Client to Server algorithms separated by a comma.
	Preferred Ciphers S2C	`sftpserver.`*`aliasName`*`.preferredCiphersS2C` Preferred Server to Client Ciphers separated by a comma.
	Preferred Ciphers C2S	`sftpserver.`*`aliasName`*`.preferredCiphersC2S` Preferred Client to Server Ciphers separated by a comma.
SFTP User Alias*	**User Name**	`sftpuser.`*`aliasName`*`.userName` The user name for the SFTP user account.
	Authentication Type	`sftpuser.`*`aliasName`*`.authenticationType` The type of authentication that Integration Server uses to authenticate itself to the SFTP server. The value can be `password` or `publicKey`.
	Private Key Location	`sftpuser.`*`aliasName`*`.privateKeyFileLocation` The location of the private key for the specified SFTP user if the authentication type is `publickey`. **Note:** The path of the private key must be relative to the installation directory.
	Password	`sftpuser.`*`aliasName`*`.password` The password for the SFTP user account.
	PassPhrase	`sftpuser.`*`aliasName`*`.passPhrase`

Asset	Property	Key Name
		The passphrase generated while creating the private key.
	SFTP Server Alias	`sftpuser.`*`aliasName`*`.sftpServerAlias` The alias of the SFTP server to which you want the SFTP user account to connect.
	Strict Host Key Checking	`sftpuser.`*`aliasName`*`.strictHostKeyChecking` Whether Integration Server verifies the host key of the SFTP server before establishing a connection to the SFTP server. Set to `yes` or `no`.
Truststore Alias*	**Description**	`truststore.`*`aliasName`*`.ksDescription` **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Type	`truststore.`*`aliasName`*`.ksType` Default supported values are: `JKS` and `PKCS12`. Other truststore types can be made available by loading additional security providers, or by setting the `watt.security.TrustStoreTypes` server configuration property. **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Provider	`truststore.`*`aliasName`*`.ksStoreProviderName` Default supported values are as follows: - If `truststore.`*`aliasName`*`.ksType` is set to `JKS`: `SUN` - If `truststore.`*`aliasName`*`.ksType` is set to `PKCS12`: `BC`, `SUN`, or `SunJSSE` **Note:** This key is available only after applying a fix that includes PIE-87505 (IS_10.11_Core_Fix15).
	Location	`truststore.`*`aliasName`*`.ksLocation`
	Password	`truststore.`*`aliasName`*`.ksPassword`
Universal Messaging	**Description**	`messaging.`*`connectionAliasName`*`.description`

Asset	Property	Key Name
Connection Alias* (webMethods messaging)		
	Client Prefix	`messaging.`*`connectionAliasName`*`.CLIENTPREFIX`
	Client Prefix Is Shared	`messaging.`*`connectionAliasName`*`.isClientPrefixShared` Whether the client prefix is shared with other Integration Servers. Set to true to indicate client prefix is shared. Set to false to indicate client prefix is not shared.
	Realm URL	`messaging.`*`connectionAliasName`*`.url`
	Maximum Reconnection Attempts	`messaging.`*`connectionAliasName`*`.um_tryAgainMaxAttempts`
	Enable CSQ	`messaging.`*`connectionAliasName`*`.useCSQ` Whether to use a client side queue for the connection alias. Set to true to use a client side queue. Set to false if you do not want the connection alias to use a client side queue.
	Maximum CSQ Size	`messaging.`*`connectionAliasName`*`.csqSize`
	Drain CSQ in Order	`messaging.`*`connectionAliasName`*`.csqDrainInOrder` Set to true to drain the client side queue in the same order in which messages were placed in the queue. Set to false if you do not want the client side queue to drain in order.
	Publish Wait Time while Reconnecting	`messaging.`*`connectionAliasName`*`.um_publishWaitTime`
	Enable Follow the Master for Producers	`messaging.`*`connectionAliasName`*`.um_followTheMasterForPublish` Set to `true` to follow the master realm server when publishing. Set to `false` to disable follow the master realm behavior for publishing.
	Enable Request-Reply Channel and Listener	`messaging.`*`connectionAliasName`*`.enableRequestReply` Set to true to use a request/reply channel for the alias. Set to false to disable use of a request/reply channel for the alias.

Asset	Property	Key Name
	Enable Follow the Master for Consumers	`messaging.`*`connectionAliasName`*`.um_followTheMasterForSubscribe` Set to true to follow the master realm server when retrieving messages. Set to false to disable follow the master realm behavior for message retrieval.
	Username	`messaging.`*`connectionAliasName`*`.user`
	Password	`messaging.`*`connectionAliasName`*`.password`
	Truststore Alias	`messaging.`*`connectionAliasName`*`.trustStoreAlias`
	Keystore Alias	`messaging.`*`connectionAliasName`*`.keyStoreAlias`
	Key Alias	`messaging.`*`connectionAliasName`*`.keyAlias`
	Logging Type	`messaging.`*`connectionAliasName`*`.um_loggingOutput` Set to `0` for SERVER LOG. Set to `1` for MESSAGING AUDIT LOG
	Enable Producer Message ID Tracking	`messaging.`*`connectionAliasName`*`.um_producerMessageTracking` Set to true to enable additional logging for message producers that use this connection alias. Set to false to disable the additional logging.
	Producer Message ID Tracking: Include Channels	`messaging.`*`connectionAliasName`*`.um_producerIncludedStrings`
	Enable Consumer Message ID Tracking	`messaging.`*`connectionAliasName`*`.um_consumerMessageTracking` Set to true to enable additional logging for message consumers (triggers) that use this connection alias. Set to false to disable the additional logging.
	Consumer Message ID Tracking: Include Triggers	`messaging.`*`connectionAliasName`*`.um_consumerIncludedStrings`
	Default Connection Alias	`messaging.`*`connectionAliasName`*`.default=true`
	Type	`messaging.`*`connectionAliasName`*`.type` Specifies whether the alias connects to Broker or Universal Messaging. Set to UM or Broker. The default is UM.
User account*	**Password**	`user.`*`userName`*`.password`

Asset	Property	Key Name
webMethods Cloud Account	**Stage**	`wmcloudaccount.`*`webMethodsCloudAccountAlias`*`. stage`
	Allowed On-Premise Hosts	`wmcloudaccount.`*`webMethodsCloudAccountAlias`*`. onPremiseHosts`
webMethods Cloud Settings	**User Name**	`wmcloudsettings.default.username`
	Password	`wmcloudsettings.default.password`
	webMethods Cloud URL	`wmcloudsettings.default.iLiveURL`
Web Service Endpoint Alias	**Host Name or IP Address**(HTTP/S Transport Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.transportInfo.host`
	Port Number (HTTP/S Transport Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.transportInfo.port`
	User Name (HTTPS Transport Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.transportInfo.user`
	Password (HTTPS Transport Properties)	`webserviceendpoint.consumer.`*`protocol`*`. `*`aliasName`*`.transportInfo.transportPassword`
	User Name (WS Security Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.messageInfo.user`
	Password (WS Security Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.messageInfo.messagePassword`
	To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.messageaddressingproperties. toMsgAddr`
	From (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.messageaddressingproperties. fromMsgAddr`
	Reply To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`. `*`aliasName`*`.messageaddressingproperties. replyToMsgAddr`
	Fault To (Message Addressing Properties)	`webserviceendpoint.`*`type`*`.`*`protocol`*`.`*`aliasName`*`. messageaddressingproperties.faultToMsgAddr`

Where *`type`* can be consumer, provider, or messageaddressing and *`protocol`* can be HTTP, HTTPS, or JMS.

C Prometheus Metrics

Prometheus Metrics Returned by Microservices Runtime

When the `metrics` endpoint is called, Microservices Runtime gathers metrics and returns the data in a Prometheus format. Prometheus is an open source monitoring and alerting toolkit which is frequently used for monitoring containers.

Microservices Runtime returns server-level metrics, service-level metrics, and JVM metrics. All of the metrics returned by Microservices Runtime are `gauge` metrics, one of the core metric types. The following sections identify the server, service-level, and JVM metrics gathered and returned by Microservices Runtime.

Microservices Runtime can also gather and return metrics for layered products. For a list and description of metrics returned by layered products, consult the documentation for those products.

Note:
The Microservices Runtime documentation assumes a familiarity with Prometheus technology. An in-depth discussion of Prometheus is beyond the scope of this guide but is available elsewhere.

Server Metrics

Server metrics are metrics about the Microservices Runtime that are returned when the `metrics` endpoint is called. Microservices Runtime obtains some metrics from MBeans available in the JVM.

For each server metric, the following table provides the name, description including the JMX MBean and the method used to obtain the metric if one was used, and the Prometheus labels.

Prometheus Metric Name	Metric Description	Prometheus Labels
`sag_is_adapter_count`	The number of adapter packages loaded on Microservices Runtime	`host`
`sag_is_average_connection_time`	The running average for the connection time expressed in milliseconds. This can also be described as the average session time or how long a connection is kept alive.	`host`
`sag_is_avg_time_per_http_requests`	The running average time for all HTTP/S requests measured in milliseconds. This metric excludes requests for services in packages listed in the	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
	watt.server.stats.packages.exclude parameter.	
`sag_is_cloudstream_count`	The number of loaded CloudStreams connectors on Microservices Runtime.	`host`
`sag_is_current_stateful_sessions`	The current number of stateful sessions.	`host`
`sag_is_custom_package_count`	The number of custom packages loaded on Microservices Runtime. A package is considered a custom package if the package manifest contains system_package=no.	`host`
`sag_is_free_memory_bytes`	The total free memory for the Microservices Runtime JVM.	`host`
`sag_is_http_requests`	The total number of HTTP/S requests since the last statistics poll. The statistics poll interval is controlled by the watt.server.stats.pollTime server configuration parameter. The default interval is 60 seconds. This metric excludes requests for services in packages listed in the watt.server.stats.packages.exclude parameter.	`host`
`sag_is_max_connections`	The peak number of connections to Microservices Runtime the connection is a session which can be created multiple ways such as through an HTTP connection, a trigger, the scheduler, etc.	`host`
`sag_is_max_memory_bytes`	The maximum memory of the Microservices Runtime JVM.	`host`
`sag_is_max_number_sessions_allowed`	The maximum number of sessions allowed at one time as determined by the license.	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
`sag_is_max_services`	The maximum number of services that have run concurrently on the server excluding services in a package set in the watt.sever.stats.packages.exclude parameter.	`host`
`sag_is_max_system_threads`	The peak number of system threads used by Microservices Runtime since startup.	`host`
`sag_is_number_current_connections`	The number of current connections to the Microservices Runtime where the connection is a session which can be created multiple ways such as through an HTTP connection, a trigger, the scheduler, etc.	`host`
`sag_is_number_nonservice_errors`	The number of errors that were caused by exceptions that are not ServiceExceptions. This is the same value returned by `sag_is_number_unknown_errors`.	`host`
`sag_is_number_service_errors`	The number of service errors since startup, which includes any service that gets an exception of any kind.	`host`
`sag_is_number_service_errors_excluding_internal_svc_per_minute`	The number of service errors in the last minute, which includes any service that gets an exception of any kind but excludes any services in a package set in the watt.sever.stats.packages.exclude server configuration parameter.	`host`
`sag_is_number_service_errors_per_minute`	The number of service errors in the last minute.	`host`
`sag_is_number_service_invokes`	The number of successful service invokes and service errors since Microservices Runtime startup. This metric excludes services in a package	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
	set in the watt.sever.stats.packages.exclude parameter.	
`sag_is_number_sessions_used`	The number of sessions used since Microservices Runtime startup.	`host`
`sag_is_number_unknown_errors`	The number of errors that were caused by exceptions that are not ServiceExceptions. This is the same value returned by `sag_is_number_nonservice_errors`.	`host`
`sag_is_peak_number_sessions`	The peak number of sessions since Microservices Runtime startup.	`host`
`sag_is_peak_service_threads`	The peak number of threads used by the server since Microservices Runtime startup.	`host`
`sag_is_peak_stateful_sessions`	The peak number of stateful sessions since Microservices Runtime startup.	`host`
`sag_is_request_duration`	The total duration in milliseconds for all service requests over the last interval where the interval length is determined by the value of the watt.server.stats.pollTime server configuration parameter. This metric excludes services in a package set in the watt.sever.stats.packages.exclude parameter.	`host`
`sag_is_server_jvm_thread_count`	Number of threads running in the Microservices Runtime JVM.	`host`
`sag_is_server_open_files_count`	Total open file descriptors. Returned only for UNIX\Linux operating systems. **JMX MBean/Method:** com.sun.management. UnixOperatingSystemMXBean. getMaxFileDescriptorCount	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
`sag_is_server_proc_cpu_percent`	Percentage of the CPU used for the Microservices Runtime JVM. **JMX MBean/Method:** com.sun.management. OperatingSystemMXBean. getProcessCpuLoad converted to an integer from 0 to 100.	`host`
`sag_is_server_proc_sys_percent`	Percentage of the CPU used by the Operating System. **JMX MBean/Method:** com.sun.management. OperatingSystemMXBean. getSystemCpuLoad converted to an integer from 0 to 100.	`host`
`sag_is_server_ session_licensed_count`	Number of active licensed Microservices Runtime sessions. (Administrator sessions are not included in this count.)	`host`
`sag_is_server_ session_stateful_count`	Number of active stateful sessions.	`host`
`sag_is_server_ session_stateless_count`	Number of active stateless sessions.	`host`
`sag_is_server_startup_Time`	The time at which Microservices Runtime started up.	`host`
`sag_is_server_sysload_average`	Number of Microservices Runtime server threads waiting for CPU resources. Returned only for UNIX\Linux operating systems. **JMX MBean/Method:** java.lang.managment .OperatingSystemMxBean .getSystemLoadAverage	`host`
`sag_is_server_total_disk_ mbytes`	Total available disk space measured in megabytes on the disk where Microservices Runtime is installed.	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
	JMX MBean/Method: java.io.File.getTotalSpace	
`sag_is_server_total_memory_mbytes`	Total amount of physical memory available on the machine on which Microservices Runtime is installed measured in megabytes. **JMX MBean/Method:** java.lang.Runtime.totalMemory	`host`
`sag_is_server_used_disk_mbytes`	Used disk space measured in megabytes on the disk where Microservices Runtime is installed. **JMX MBean/Method:** The value of the metric is calculated by subtracting the value obtained from java.io.File.getFreeSpace from java.io.File.getTotalSpace.	`host`
`sag_is_server_used_memory_mbytes`	Total amount of physical memory used on the machine on which Microservices Runtime is installed measured in megabytes measured in megabytes. **JMX MBean/Method:** The value of the metric is calculated subtracting the value of java.lang.Runtime.freeMemory from java.lang.Runtime.totalMemory.	`host`
`sag_is_service_completions_per_minute`	The number of top-level service completions per minute.	`host`
`sag_is_service_count`	The number of services in custom packages loaded on Microservices Runtime. A package is considered a custom package if the package manifest contains system_package=no.	`host`
`sag_is_service_starts_per_minute`	The number of top-level services started each minute.	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
`sag_is_service_threads`	The total number of threads used for service execution where the threads are obtained from the server thread pool	`host`
`sag_is_services`	The total number of running services currently active excluding services in a package set in the watt.sever.stats.packages.exclude parameter.	`host`
`sag_is_stateful_sessions_limit`	The maximum number of stateful sessions. If watt.server.session.stateful.max is set to 0 or watt.server.session.stateful.enableLimit is set to false, this is the maximum number of sessions as specified in the license. If watt.server.session.stateful.max is not 0 and watt.server.session.stateful.enableLimit is set to true, then this is the value of watt.server.session.stateful.max.	`host`
`sag_is_system_package_count`	The number of loaded system packages on Microservices Runtime. A package is considered a system package if the package manifest contains system_package=yes.	`host`
`sag_is_system_threads`	The total number of Java system threads which is any thread not obtained from the server thread pool.	`host`
`sag_is_total_http_requests`	The total number of HTTP/S requests since Microservices Runtime startup. This metric excludes requests for services in packages listed	`host`

Prometheus Metric Name	Metric Description	Prometheus Labels
	in the watt.server.stats.packages.exclude parameter.	
`sag_is_total_memory_bytes`	The total memory for the Microservices Runtime JVM.	`host`
`sag_is_total_request_durations`	The total duration, in milliseconds, for all service requests since Microservices Runtime startup measured in milliseconds. This metric excludes services in a package set in the watt.sever.stats.packages.exclude parameter.	`host`
`sag_is_used_memory_bytes`	The total used memory for the Microservices Runtime JVM.	`host`

Service Metrics

Service metrics are metrics about a service on Microservices Runtime. When the metrics endpoint is called, Microservices Runtime returns service metrics for services called as top-level services. Microservices Runtime does not return metrics for internal services or nested services which are those services invoked within another service.

Note:
A top-level service is a service that is invoked by a client request.

The following table identifies the service metrics returned by Microservices Runtime when the `metrics` endpoint is called, a description of the metric, and the Prometheus label for the metric.

Prometheus Metric Name	Description and Prometheus Label	Prometheus Label
`sag_is_service_cache_ entries`	The current number of active cached entries for this service in service results cache.	`host, service`
`sag_is_service_cache_ expires`	The current number of expired cache entries for the service.	`host, service`
`sag_is_service_cache_ hit_ratio`	The percentage of request for the service that have been fulfilled using cached service	`host, service`

Prometheus Metric Name	Description and Prometheus Label	Prometheus Label
	results since the cache was last reset.	
`sag_is_service_number_ access`	The number of times the service has executed since the Microservices Runtime last started.	`host, service`
`sag_is_service_number_ cache_hit`	The total number of times that a cached service result pipeline has been returned instead of calling the service since the cache was last reset.	`host, service`
`sag_is_service_number_ circuit_open`	The number of times the circuit has opened for this service since Microservices Runtime last started. This metric applies only to services for which a circuit breaker is enabled.	`host, service`
`sag_is_service_number_ circuit_open_request`	The number of requests for the service when the circuit was in an open state since Microservices Runtime last started. This metric applies only to services for which a circuit breaker is enabled.	`host, service`
`sag_is_service_number_errors`	The total number of errors for the service since Microservices Runtime last started.	`host, service`
`sag_is_service_number_ prefetch`	The number of times that Microservices Runtime has prefetched and then cached the results for this service since the cache was last reset.	`host, service`
`sag_is_service_number_ recent_prefetch`	The number of times Microservices Runtime has prefetched and cached the results for the service since the last statistics poll. The statistics poll interval is controlled by the watt.server.stats.pollTime server	`host, service`

Prometheus Metric Name	Description and Prometheus Label	Prometheus Label
	configuration parameter. The default interval is 60 seconds.	
`sag_is_service_number_running`	The number of concurrent service executions for this service at the time Microservices Runtime gathered the metrics.	`host, service`
`sag_is_service_requests_avg_exec_millis`	The average number of milliseconds to process the Microservices Runtime service in the last polling interval.	`apiCat, code, execStat, host, service, origin`
`sag_is_service_requests_total`	The total number of times the service was executed in the last polling interval.	`apiCat, code, execStat, host, origin, service,`

JVM Metrics

JVM metrics are statistics about the JVM in which Microservices Runtime runs. The JVM contains a wide variety of information which is exposed via Managed Beans (MBeans). Microservices Runtime gathers the JVM metrics data from available MBeans.

Note:
The JDK internal MBeans belong to the java.lang.management package.

The following table identifies the JVM metrics returned by the `metrics` endpoint, a description of each metric, the Prometheus label for the metric, and the JMX MBean and method from which Microservices Runtime obtains the data.

Prometheus Metric Name	Description	Prometheus Label
`sag_is_jvm_classes_loaded_total`	Total number of current classes loaded in the Microservices Runtime JVM. **JMX MBean/Method:** java.lang.management. ClassLoadingMXBean. getLoadedClassCount	`host`
`sag_is_jvm_classes_total`	The total number of classes loaded in the Microservices Runtime JVM since it was started.	`host`

Prometheus Metric Name	Description	Prometheus Label
	JMX MBean/Method: java.lang.management. ClassLoadingMXBean. getTotalLoadedClassCount	
`sag_is_jvm_classes_ unloaded_total`	Total number of classes unloaded in the Microservices Runtime JVM. **JMX MBean/Method:** java.lang.management. ClassLoadingMXBean. getUnloadedClassCount	`host`
`sag_is_jvm_gc_collection_count`	The total number of freed objects from garbage collection cycle. **JMX MBean/Method:** java.lang.management. GarbageCollectorMXBean. getCollectionCount	`host, name`
`sag_is_jvm_gc_collection_millis`	Number of milliseconds elapsed since the last garbage collection cycle. **JMX MBean/Method:** java.lang.management. GarbageCollectorMXBean. getCollectionTime	`host, name`
`sag_is_jvm_memory_buffer_ pool_capacity_mbytes`	The total number of allocated megabytes from NIO buffer pools. **JMX MBean/Method:** java.lang.management. BufferPoolMXBean.getTotalCapacity	`host, name`
`sag_is_jvm_memory_ buffer_pool_count`	The total number of buffers from NIO buffer pools. **JMX MBean/Method:** java.lang.management. BufferPoolMXBean.getCount	`host, name`

Prometheus Metric Name	Description	Prometheus Label
`sag_is_jvm_memory_buffer_pool_used_mbytes`	The total number of used megabytes from NIO buffer pools. **JMX MBean/Method:** java.lang.management. BufferPoolMXBean.getMemoryUsed	`host, name`
`sag_is_jvm_memory_heap_used_mbytes`	The total number of allocated megabytes from heap memory in the Microservices Runtime JVM. **JMX MBean/Method:** java.lang.management. MemoryMXBean.getHeapMemoryUsage	`host`
`sag_is_jvm_memory_nonheap_used_mbytes`	The total number of allocated megabytes that are not from heap memory in the Microservices Runtime JVM. **JMX MBean/Method:** java.lang.management. MemoryMXBean. getNonHeapMemoryUsage	`host`
`sag_is_jvm_memory_pool_committed_mbytes`	The total number of committed megabytes in the named memory pool. **JMX MBean/Method:** java.lang.management. MemoryPoolMXBean java.lang.management. MemoryUsage.getCommitted()	`host, name, poolType`
`sag_is_jvm_memory_pool_init_mbytes`	The total number of initially allocated megabytes in the named memory pool. **JMX MBean/Method:** java.lang.management. MemoryPoolMXBean java.lang.management. MemoryUsage.getInit()	`host, name, poolType`

Prometheus Metric Name	Description	Prometheus Label
`sag_is_jvm_memory_pool_max_mbytes`	The total number of allocated megabytes in the named memory pool. **JMX MBean/Method:** java.lang.management.MemoryPoolMXBean java.lang.management.MemoryUsage.getMax()	`host, name, poolType`
`sag_is_jvm_memory_pool_used_mbytes`	The total number of used megabytes in the named memory pool. **JMX MBean/Method:** java.lang.management.MemoryPoolMXBean java.lang.management.MemoryUsage.getUsed()	`host, name, poolType`
`sag_is_jvm_objects_pending_finalizer_count`	The number of objects ready for garbage collection. **JMX MBean/Method:** java.lang.management.MemoryMXBean.getObjectPendingFinalizationCount	`host`
`sag_is_jvm_thread_state_blocked_count`	The number of BLOCKED threads where BLOCKED is defined in java.lang.Thread.State. **JMX MBean/Method:** java.lang.management.ThreadMXBean java.lang.management.ThreadInfo.getBlockedCount	`host`
`sag_is_jvm_thread_state_new_count`	The number of NEW threads where NEW is defined in java.lang.Thread.State. **JMX MBean/Method:** java.lang.management.ThreadMXBean	`host`
`sag_is_jvm_thread_state_runnable_count`	The number of RUNNABLE threads where RUNNABLE is defined in java.lang.Thread.State.	`host`

Prometheus Metric Name	Description	Prometheus Label
	JMX MBean/Method: java.lang.management.ThreadMXBean	
sag_is_jvm_thread_ state_terminated_count	The number of TERMINATED threads where TERMINATED is defined in java.lang.Thread.State. **JMX MBean/Method:** java.lang.management. ThreadMXBean.dumpAllThreads	host
sag_is_jvm_thread_ state_timed_waiting_count	The number of TIMED_WAITING threads with a timeout where TIMED_WAITING is defined in java.lang.Thread.State. **JMX MBean/Method:** java.lang.management.ThreadMXBean	host
sag_is_jvm_thread _state_waiting_count	The number of WAITING threads without a timeout where WAITING is defined in java.lang.Thread.State. **JMX MBean/Method:** java.lang.management.ThreadMXBean	host
sag_is_jvm_threads _blocked_count	The number of blocked threads. **JMX MBean/Method:** java.lang.management.ThreadMXBean	host
sag_is_jvm_ threads_blocked_millis	The total number of milliseconds threads have been in a blocked state. **JMX MBean/Method:** java.lang.management.ThreadMXBean java.lang.management. ThreadInfo.getBlockedTime	host
sag_is_jvm_threads_ native_count	The number of threads executing code outside of the JVM. **JMX MBean/Method:** java.lang.management.ThreadMXBean	host

Prometheus Metric Name	Description	Prometheus Label
	java.lang.management. ThreadInfo.isInNative	
`sag_is_jvm_` `threads_suspend_count`	The number of suspended threads. **JMX MBean/Method:** java.lang.management.ThreadMXBean java.lang.management. ThreadInfo.isSuspended	`host`
`sag_is_jvm_` `threads_waited_count`	The number of times the thread has been blocked. **JMX MBean/Method:** java.lang.management.ThreadMXBean java.lang.management. ThreadInfo.getWaitedCount	`host`
`sag_is_jvm_` `threads_waited_millis`	Total number milliseconds all threads have waited for a lock. **JMX MBean/Method:** java.lang.management.ThreadMXBean java.lang.management. ThreadInfo.getWaitedTime	`host`

Prometheus Labels

Prometheus metrics can contain labels which can be used with a metric to differentiate metrics returned by the `metrics` endpoint from each other. For example the label `service="createCustomer"` used with the metric `sag_is_service_requests_total` indicates that the metric describes the total number of requests made in the last polling interval for the service createCustomer. Whereas the label `service="updateCustomer"` used with the same metric indicates the total number of requests made in the last polling interval for the service updateCustomer.

The following table identifies the Prometheus labels that may be associated with a Prometheus metric returned by the `metrics` endpoint.

Prometheus Label	Description
`apiCat`	API Category for a service invocation. ■ API_GRAPHQL ■ API_ODATA

Prometheus Label	Description
	■ API_REST ■ API_SOAP ■ API_WEBSOCKET (Outbound service calls to a remote WebSocket listener only.)
`code`	An integer value associated with the metric. The code relates to the type of service that is executing. For example, a service that receives an HTTP response will have an HTTP status code.
`execStat`	The execution state of the service. The `execStat` can be one of the following values: ■ Y for service success. ■ N for service failure. ■ U for unknown which indicates the service had not yet executed completion at the time the metrics were gathered.
`host.`	Host name of theMicroservices Runtime in the format: *<hostname|ipaddress>* .
`name`	Name associate with the metric. The name value varies and depends on the data in the metric. For example, if the metric is for a service, the name might be the service name. If the metric is for a memory-related metric, the name might be the memory pool name.
`origin`	Origin for the service invocation. The type of origin information will vary per `apiCat`. For example for a service invocation for an API_REST call, the origin is the REST resource endpoint. For an API_SOAP call, the origin is a service endpoint.
`poolType`	Type of memory pool as returned by java.lang.management.MemoryPoolMXBean.getType. The type name varies depending on the JDK implementation.
`service`	Fully qualified name of a service.